Application of the Homotopy Continuation Method to Low-Eccentricity Preliminary Orbit Determination

by

Andrew Markland Stanley Hart

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1991

Author .....
Department of Aeronautics and Astronautics
June, 1991

Certified by .....
Richard Battin
Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by .....
David W. Carter
Staff Member, Draper Laboratory
Thesis Supervisor

Accepted by .....
Professor Harold Y. Wachman
Chairman, Departmental Committee on Graduate Students

Application of the Homotopy Continuation Method to Low-Eccentricity Preliminary Orbit Determination

by

Andrew Markland Stanley Hart

Submitted to the Department of Aeronautics and Astronautics
on June, 1991, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

The homotopy continuation method is a method for solving a set of non-linear equations.

This thesis examines the application of the homotopy continuation method to the preliminary orbit determination of satellites with low eccentricity orbits, extending and modifying techniques of R. L. Smith and C. Huang. Further improvements to the method, to allow more general orbit determination, are considered.

In particular this thesis deals with orbit determination using only six range measurements from a single ground-based tracking station. Application to the Landsat 6 satellite, which is due to be launched in May 1992, is considered.

Thesis Supervisor: Richard Battin
Title: Professor of Aeronautics and Astronautics

Thesis Supervisor: David W. Carter
Title: Staff Member, Draper Laboratory

Acknowledgments

I would like to thank David Carter of The Charles Stark Draper Laboratory for the invaluable time and assistance he provided over the last eight months.

I would also like to thank the Massachusetts Institute of Technology for the financial aid provided by it during that time.

A final note of thanks goes to Professor Richard Battin for his instruction in courses 16.46 and 16.47, which were both interesting and directly applicable to this work, for providing me with the opportunity to work on this subject and for acting as my thesis supervisor.

Biographical Note

The author graduated with first class honors in June 1989 from the University of Southampton, England, with a Bachelor of Engineering degree in Aeronautical and Astronautical Engineering.

While at The Massachusetts Institute of Technology the author spent six months as a research assistant doing experimental work in The Wright Brothers Wind Tunnel.

Contents

List of Figures

Chapter 1

Introduction

The purpose of any satellite orbit determination method is to use tracking data to determine the orbit of a satellite at some epoch time so that predictions regarding the future path of that satellite can be made. This research addresses the use of the homotopy continuation method with only six range measurements from a single Earth based tracking station for orbit determination. A range measurement is simply a measurement of the distance between the tracking station and the satellite.

1.1 Motivation

Up until recently the Landsat program was owned and operated by the United States government (NASA). The tracking data available to the program consisted of range, range-rate and angle data from a number of different Earth tracking stations, as well as data from the Tracking and Data Relay Satellite System (TDRSS).

In 1984, the Landsat Remote Sensing Commercialization Act was passed by Congress. This act allowed a contract for the operation and expansion of the Landsat system to be awarded to a private firm through competitive bidding. The contract was awarded in September 1985 to the Earth Observation Satellite Company (EOSAT) which is based in Lanham, Maryland.

EOSAT is a partnership formed in 1984 by General Motors' Hughes Aircraft Company, the General Electric Astro-Space Division, and Computer Sciences Corporation.

EOSAT is responsible for the development of the new Landsat and for worldwide marketing of Landsat data. It has initiated the development of the next generation Landsat satellite, Landsat 6.

EOSAT currently receives ephemeris and payload data from the Goddard Space Flight Center(GSFC) in Greenbelt, Maryland, and through the Tracking and Data Relay Satellite System. Tracking data consisting of range, range-rate, and angle data is processed at NASA GSFC. This means that EOSAT does not have control over what data they receive. Also, they have to pay for that data. For these reasons, EOSAT is planning to replace the NASA and other U.S. Government facilities with its own facilities for Landsat 6. This will include a tracking and data reception station in Norman, Oklahoma.

Range-rate is measured by observing the Doppler frequency shift between the satellite transponder and the tracking station receiver. The planned Landsat 6 transponder will be operating at frequencies which experience a maximum of about only 2 Hz Doppler shift horizon-to-horizon [10]. This means that range-rate measurements would have poor resolution. EOSAT plans to have some angle measurement equipment in the tracking station, but angle measurements are sensitive to atmospheric refraction, especially when the satellite is low over the horizon, and so may not be accurate enough to use for preliminary orbit determination. A preliminary orbit determination method that requires only range measurements seems desirable. An in depth discussion of the history and functions of Landsat and EOSAT is contained in reference [11].

Out of the desire for a preliminary orbit determination method which requires only range measurements from a single Earth-based tracking station comes the motivation for the adaptation of the homotopy continuation method presented in this thesis.

1.2 Previous Work Concerning Preliminary Orbit Determination

A variety of techniques using varying combinations of range, range-rate and angle data have been researched in the past. A number of the classical orbit determination methods are presented in reference [9]. Chapter 9 of reference [8] presents algorithms using range and angle data. In reference [13] a preliminary orbit determination method using satellite-to-satellite range and range-rate data is presented. Reference [14] presents a method requiring range-only data from a single station but the method only applies to circular orbit determination. Another range-only method is presented in reference [15], but the method is a least squares method and so has a limited radius of converegence, and it also uses satellite-to-satellite tracking.

1.3 Previous Work Concerning The Homotopy Continuation Method

Many of the ideas for this research were derived from a National Aeronautics and Space Administration technical memorandum by R. L. Smith and C. Huang [1]. That memorandum addressed a similar early orbit determination problem, based on the use of range and range-rate observational data as obtained from the Tracking and Data Relay Satellite System (TDRSS). In this work, only range data obtained from a single Earth-bound tracking station is considered.

A more condensed and easier to read version of the technical memorandum mentioned above can be found in an American Institute of Aeronautics and Astronautics paper by the same authors [2].

A theoretical basis for the homotopy continuation method is presented in reference [3]. Examples of the homotopy continuation method applied to several engineering problems, not including orbit determination, are provided in reference [6].

1.4 Outline of Thesis

This chapter presents some of the practical background that led to this particular piece of research and provides information about related research. Chapter 2 presents a description of the homotopy continuation method as applied to preliminary orbit determination. Chapter 3 describes each of the steps in the algorithm designed to solve the early orbit determination problem. Chapter 4 presents a detailed discussion of the Newton-Raphson corrector step which is the major computational step in the algorithm. Chapter 5 discusses the problems associated with this method of preliminary orbit determination. Chapter 6 contains a discussion of the trick used to overcome the singularity at inclination $\approx 63.5^{\circ}$ in the Brouwer-Lyddane orbit propagator. Chapter 7 contains a sample of the results produced by running the program on some real Landsat 4 data. Chapter 8 presents a good strategy for using the software developed during this research. Chapter 9 provides a list of the conclusions drawn from this research. In Chapter 10, recommendations for how this work could be extended and improved are suggested. Appendix A contains a brief description of the Gaussian quadrature technique for evaluating integrals. Appendix B contains a user's guide, provides information about the computer subroutines used in this research, and also contains complete source code of the main program and all of the sub-programs.

Chapter 2

The Method

2.1 Setting Up the Problem

The problem is to carry out orbit determination of artificial satellite using only six range (distance between the tracker and the satellite) measurements from a single Earth based tracker. The method that is employed is the homotopy continuation method.

A clear and simple statement of the objective of this method is as follows :
Given the following data :

- the Earth-fixed coordinates of a single tracking station
- the right ascension of Greenwich at some epoch time
- six fixed times relative to the epoch time
- six range observations at those six fixed times relative to epoch
- an a priori estimate to the satellite state at epoch
- an orbit state propagation model, that is, some orbital dynamics model such as two-body mechanics

Calculate the exact satellite state at epoch. In this thesis a satellite state consists of a set of six orbital elements.

Let a general epoch orbit state be represented by $\mathbf{x}$, a general set of six ranges at the six observation times be represented by $\mathbf{y}$ and the function that maps the epoch state to the ranges be represented by $\mathbf{f}$. This function includes an orbit propagator which propagates the epoch state to a current state at each of the six chosen times. Then those current states are converted to inertial Cartesian coordinates. The tracking station's position in inertial Cartesian coordinates, at each of the times, is also computed. Finally, the root sum of the squares is used to compute the ranges from the satellite and station positions at each of the six times. Note that $\mathbf{f}$ depends implicitly on the six times chosen and on the propagation model.

The a priori estimate of the epoch state is denoted by $\mathbf{x}_{est}$, the set of ranges calculated by evaluating $\mathbf{f}(\mathbf{x}_{est})$ is denoted by $\mathbf{y}_{est}$ and the observed set of ranges is denoted by $\mathbf{y}_{ex}$. Then, we can say that by definition :

$$\mathbf{f}(\mathbf{x}_{est}) = \mathbf{y}_{est} \tag{2.1}$$

and the problem may be restated as a search for the exact epoch state $\mathbf{x}_{ex}$ that solves the equation :

$$\mathbf{f}(\mathbf{x}_{ex}) = \mathbf{y}_{ex} \tag{2.2}$$

Now consider the general set of ranges $\mathbf{y}$ given by :

$$\mathbf{y} = \lambda \mathbf{y}_{ex} + (1 - \lambda)\, \mathbf{y}_{est} \tag{2.3}$$

where λ is the homotopy parameter.

Consider also, the locus of solutions (λ,$\mathbf{x}$) to the equation :

$$\mathbf{f}(\mathbf{x}) = \lambda \mathbf{y}_{ex} + (1 - \lambda)\, \mathbf{y}_{est} \tag{2.4}$$

Clearly, for $\lambda = 0$, $\mathbf{x}_{est}$ is a solution and for $\lambda = 1$, $\mathbf{x}_{ex}$ is a solution. Note however that for any given value of λ the number of solutions to equation 2.4 may be zero, one, or more.

The assumption of the homotopy continuation method is that the locus of solutions

to equation 2.4 in the seven dimensional λ-$\mathbf{x}$ space, is a set of one or more disjoint, smooth, closed loops. A very simple example of a possible solution curve is shown in figure 2-1.

From the figure it is evident that in this simple case there is one solution at $\lambda = 1$, one solution at $\lambda = 0$, two solutions for all λ in the range $0 < \lambda < 1$, and no solutions with $\lambda < 0$ or with $1 < \lambda$.

It is constructive at this point to consider the effect on the shape of the solution curve(s), of changing the a priori estimated epoch state, $\mathbf{x}_{est}$, while keeping the remainder of the problem parameters constant.

Logically, a different a priori estimate should have no effect whatsoever on the actual solution states, i. e. , the $\lambda = 1$ solutions. This is borne out by the fact that the set of ranges $\mathbf{y}$ given by equation 2.3 for $\lambda = 1$ is always $\mathbf{y}_{ex}$ and so is independent of $\mathbf{x}_{est}$. For all other values of λ we would expect solutions to change as $\mathbf{x}_{est}$ is changed, and so the overall solution curve should change shape. Furthermore, we should expect that if the a priori estimate is varied smoothly, so too would the shape of the solution curve. Figure 2-2 shows an example of this taking place.

This figure serves to illustrate an additional point : as the a priori estimate varies, the solution curve can smoothly separate into two loops and in fact, for some special a priori estimate, the solution curve will be at a transition point between being one loop with two lobes and being two separate loops. The significance of this type of transition point will be discussed later on in this thesis after the mechanics of the method have been addressed, something which will be dealt with in the very next section.

2.2 Mechanics of the Method

Theoretically, the method simply entails following the solution curve in the seven dimensional $(\lambda, \mathbf{x})$ space around from the a priori state, which is a point at $\lambda = 0$ on the curve, through all the $\lambda = 1$ points(candidate solutions), recording them as we pass, back to initial start point at $\lambda = 0$. It may be necessary to screen the solutions

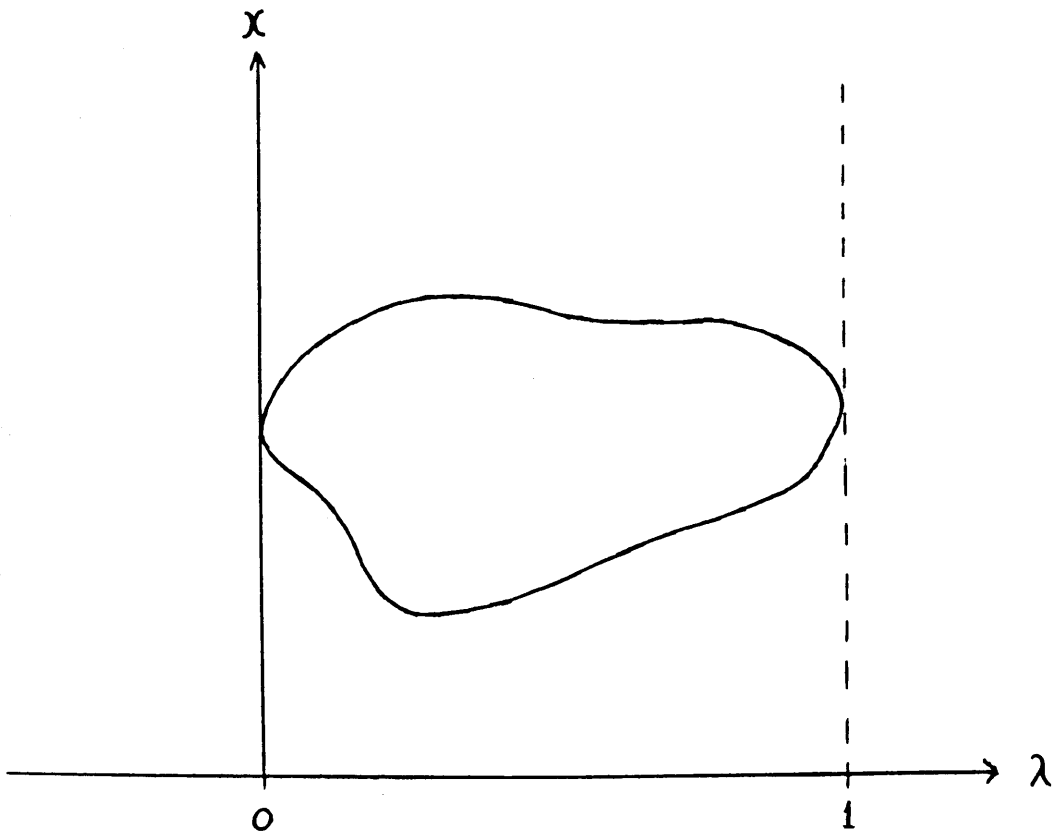

Figure 2-1: Schematic Diagram of a Simple Solution Curve

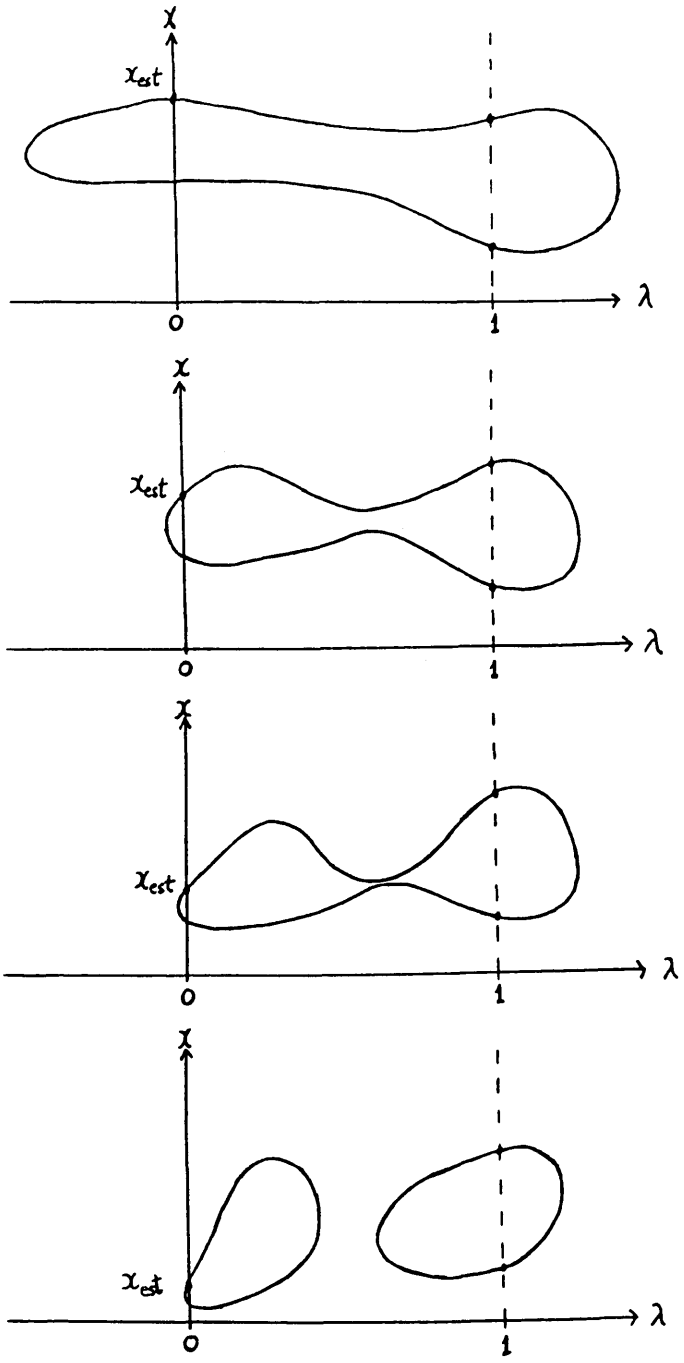

Figure 2-2: Schematic Diagram Showing the Possible Effects of Varying the A Priori Estimate on the Shape of the Solution Curve

in some way in order to pick out the solution which we seek. That is, the solution which agrees with existing a priori knowledge and which is sufficiently accurate to be useful for making good predictions about the satellite state at future times.

At this time, it should be pointed out that in a case where more than one disjoint loop exists there is no guarantee that your a priori estimate will lie on the same loop as the solution which you seek [see figure 2-3]. However, it was found that the more accurate the a priori guess, the more likely it will be that that guess and the solution which you seek will lie on the same loop. What needs to be done if this ideal situation does not exist will be dealt with later on in this thesis. For now, it is constructive to assume that a good a priori estimate exists and so the a priori estimate and the solution which you seek do lie on the same loop. For the Landsat 6 case this a reasonable assumption because it will be launched southwards from the Western Test Range (WTR) in California and so its ascending node will have longitude very close to $180°+$ the right ascension of WTR (which depends on the time of launch). Also, its inclination will be accurately known ($\approx 98°$).

The curve following scheme is a simple predictor–corrector scheme. It consists of using three points on the loop that have already been calculated, to predict what another point further along the loop will be, and then to correct that prediction using a Newton-Raphson iterative scheme. Figure 2-4 illustrates this scheme.

The prediction is made according to quadratic Lagrange extrapolation of the solution curve by some previously determined curve length. There is a problem at the start because we only have one known point initially, that is, our a priori estimate. Thus, a special starter scheme is needed for calculating the second point along the loop. Once the second point has been found, those two points can be used to make a prediction based on linear extrapolation, which can then be corrected to give a third point. After each point is corrected, there are a number of procedures that need to be carried out before the next prediction step. A summary of the overall algorithm is shown below :

starter Implement starter scheme to get the second point along the loop and calculate the arc length between the two points on the loop.

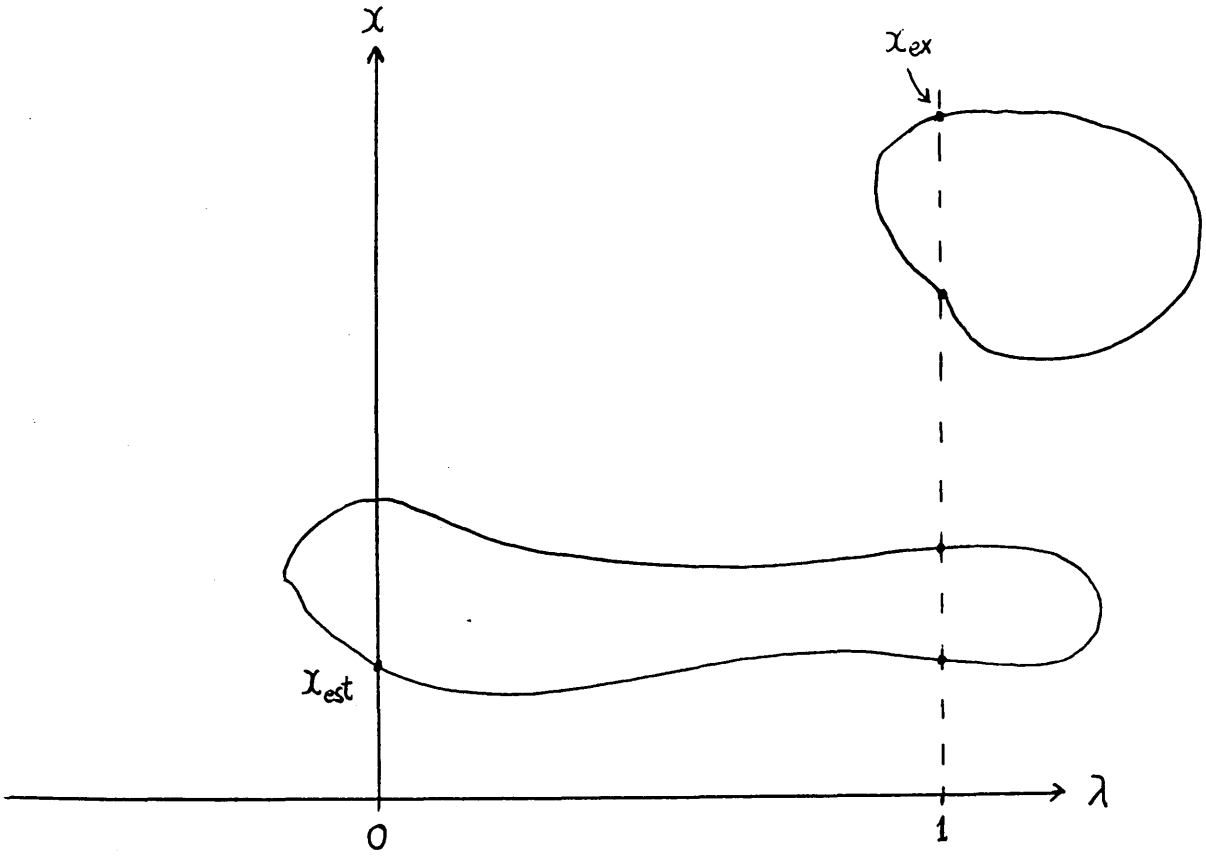

Figure 2-3: Schematic Plot Showing a Case Where the A Priori Estimate and the Solution Which We Seek Lie on Different Loops

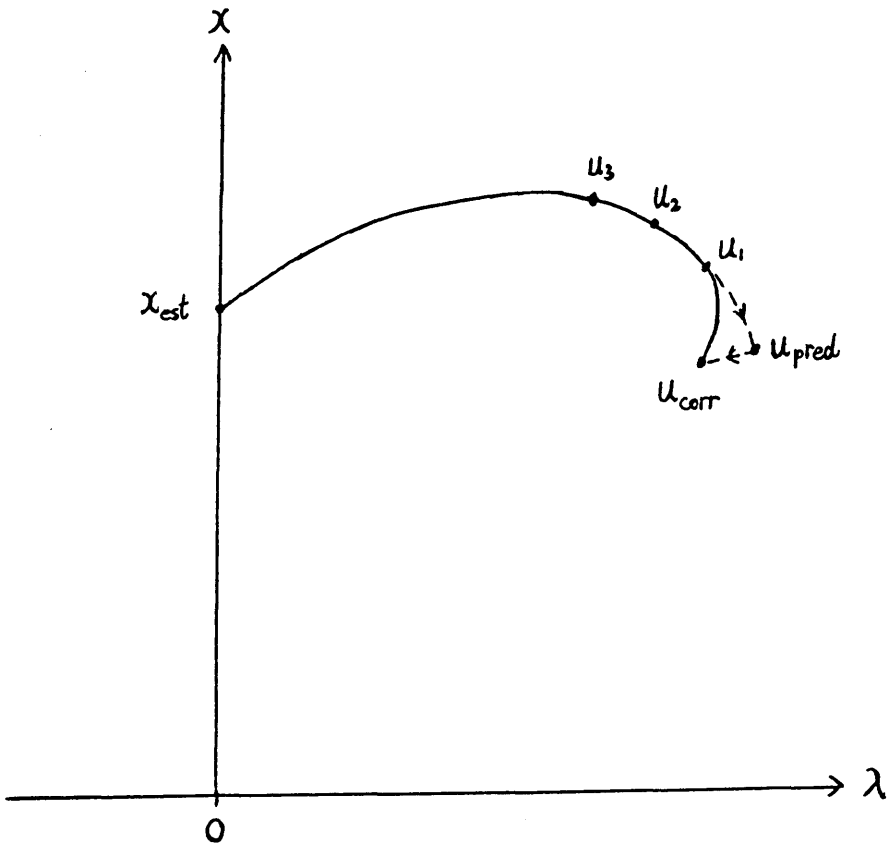

Figure 2-4: Schematic Diagram Illustrating the Corrector Scheme Employed

step size selector Calculate the step-size (in terms of arc length) that should be taken between the last point and the next point along the curve.

predictor Predict the next point using an extrapolation of the existing points and the step-size calculated.

corrector Correct that prediction using a Newton-Raphson iterative scheme with the prediction as an initial guess.

monitor Ensure that the corrector scheme converged and that the tangent vector at each of the last two corrected points do not differ by much. If either of these is not true, then return to the previous predictor step with a step-size that is halved.

arclength corrector Correct the arc length between the last two corrected points because during the transition from predicted point to corrected point the arc length may have changed from the initial step-size.

solution state collector Check for and store any crossings of the $\lambda = 1$ hyperplane (candidate solutions)

critical point collector Check for and store any points that are extrema in λ. Such points are called critical points.

terminator Check for a return to the initial start point to see if the loop has been completed. If it has, terminate the algorithm. If not, return to step 2.

A more in depth description of each of these algorithm steps is the subject of the next chapter.

2.3 Direct Newton-Raphson

The more simplistic approach of applying the Newton-Raphson method directly to the non-linear equations is not satisfactory. One reason is that the Newton-Raphson

method has a very limited radius of convergence. That is, even a good a priori estimate can result in the Newton-Raphson scheme diverging. The second reason is that the problem can have several solutions and so even if the Newton-Raphson scheme converges it may converge to a spurious solution. That is why we need a method that collects all of the possible solutions and then screens out the solution which we seek. Figure 2-5 illustrates a case where the simple Newton-Raphson scheme converges to the wrong solution. Figure 2-6 illustrates a case where the simple Newton-Raphson scheme diverges.

An interesting study of the application of the Newton-Raphson method applied directly to the Landsat 6 problem is presented in reference [16].

2.4 Screening

The screening method would depend on how much a priori information was available about the satellite. For example, some of the solutions may be ruled out by the fact that the amount of energy associated with that state may be greater than the maximum amount of energy that the launch vehicle was capable of imparting to the satellite. Possibly some of the solutions may have inclinations or ascending nodes outside of the limits within which the real inclination or ascending node is known to lie from a priori knowledge. Other solutions may be ruled out by the rough angle measurements made at the tracking station. However, even if there was absolutely no prior knowledge about the vehicle, we could use redundant observations to screen out the solution which we seek. In the case of Landsat orbit determination we will have good a priori knowledge and so we would not have to resort to using redundant observations as a screening method.

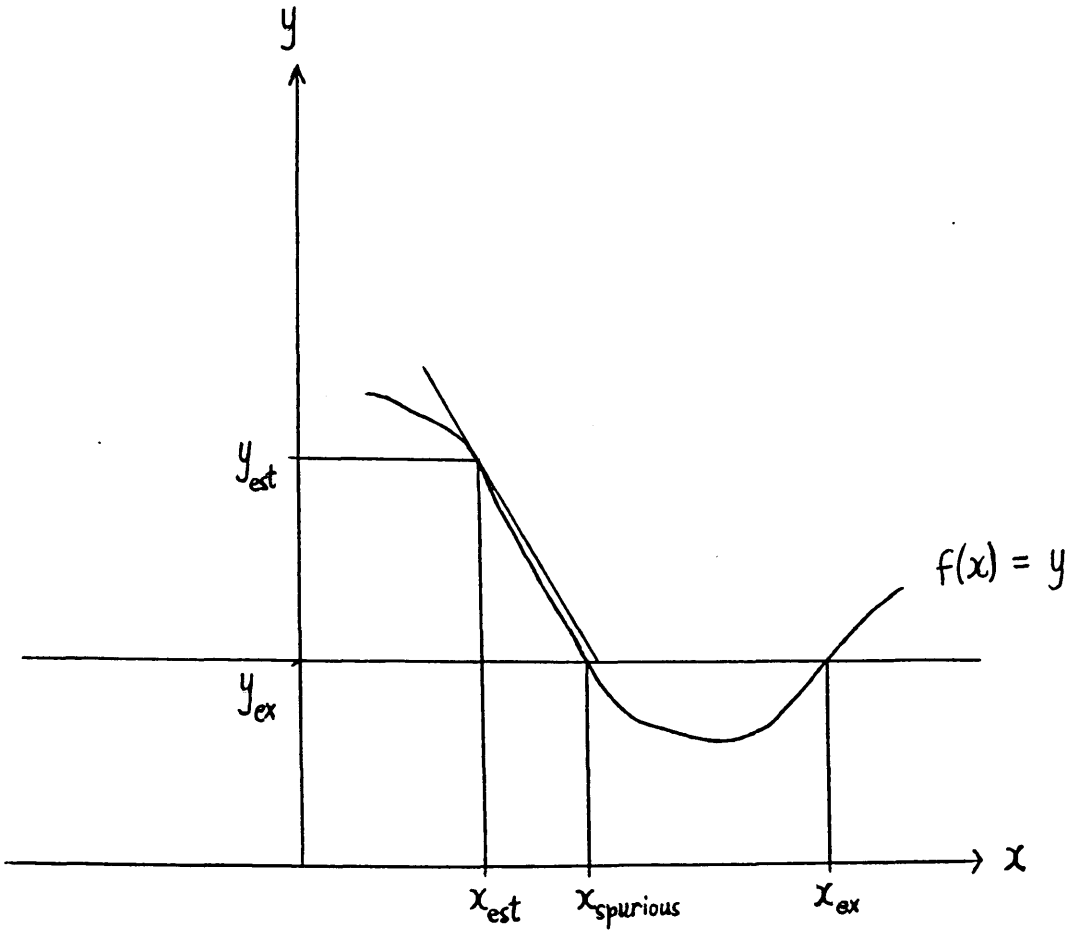

Figure 2-5: Schematic Diagram of a Case Where the Simple Newton-Raphson Method May Converge to the Wrong Solution

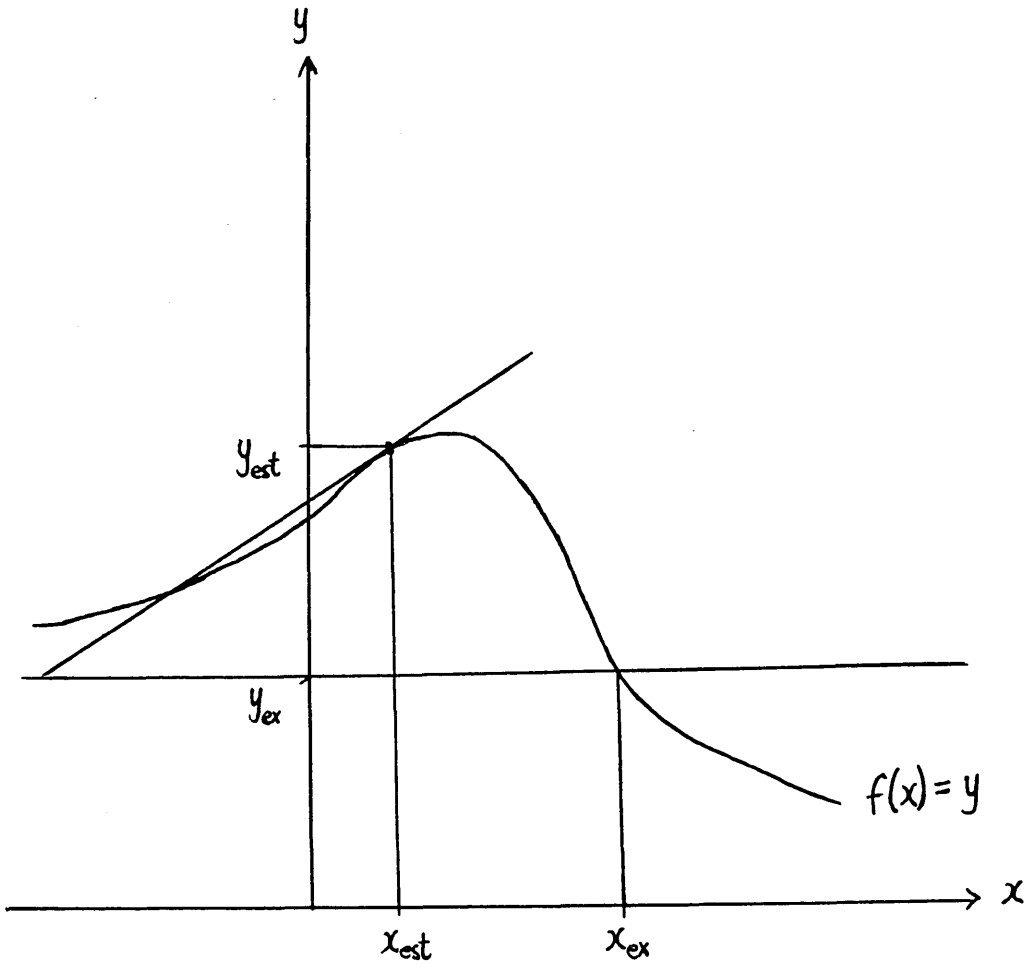

Figure 2-6: Schematic Diagram of a Case Where the Simple Newton-Raphson Method May Diverge

Chapter 3

Curve Following Algorithm

This chapter describes each of the steps in the algorithm in some detail, noting in particular any differences from the Smith paper [1], and giving reasons for those differences. The first difference to note is that Smith defined his orbit state in terms of position and velocity coordinates, while for this thesis, orbit states are defined in terms of orbital elements.

The algorithm is divided into nine steps, a starter step and then eight other steps which are repeated cyclically until the loop is completed. A good way to regard the starter is to consider it as the first cycle of eight steps ; some of the steps in this cycle are left out because they are unnecessary and others are done differently because there is very limited information at the start.

A little notation is constructive at this point. A general point in the seven dimensional curve space is denoted by $\mathbf{u}$ where $\mathbf{u} = (\lambda, \mathbf{x})$. Distance along the curve from the start point is denoted by s and step size is denoted by δs. The last corrected point on the solution curve is denoted by $\mathbf{u}_1$, the one before last corrected point on the solution curve is denoted by $\mathbf{u}_2$ and its predecessor is denoted by $\mathbf{u}_3$. A predicted curve point is denoted by $\mathbf{u}_{pred}$. The a priori point which is the first point on our solution curve is given as $\mathbf{u}_{est} = (0, \mathbf{x}_{est}) = (0, x_{est}^{(1)}, x_{est}^{(2)}, x_{est}^{(3)}, x_{est}^{(4)}, x_{est}^{(5)}, x_{est}^{(6)})$.

3.1 Starter

The technique used in the starter is as follows :

- Arbitrarily select a step size $\delta s = \delta s_{arb}$.
- Predict a solution $\mathbf{u}_{pred} = (\delta s, \mathbf{x}_{est}) = (\delta s, x_{est}^{(1)}, x_{est}^{(2)}, x_{est}^{(3)}, x_{est}^{(4)}, x_{est}^{(5)}, x_{est}^{(6)})$.
- Correct that prediction using a Newton-Raphson iterative corrector. The corrector scheme will be discussed in in the Corrector section of this chapter.
- If the iterative scheme fails, then change the prediction to

$$\mathbf{u}_{pred} = (0, x_{est}^{(1)} + \delta s, x_{est}^{(2)}, x_{est}^{(3)}, x_{est}^{(4)}, x_{est}^{(5)}, x_{est}^{(6)})$$

and try the corrector on that. If that fails, change prediction to

$$\mathbf{u}_{pred} = (0, x_{est}^{(1)}, x_{est}^{(2)} + \delta s, x_{est}^{(3)}, x_{est}^{(4)}, x_{est}^{(5)}, x_{est}^{(6)})$$

and try the corrector on that. If after going through this process of adding δs to each of the components of $\mathbf{u}_{est}$ in turn, the iterative scheme has not converged in any case, then let $\delta s = \delta s_{arb}/2$ and repeat the process. Keep halving δs until the iteration scheme converges and you have your second point on the solution curve.

Three important points to note here :

1. Assuming that the data is correct, the starter cannot fail although it may require δs to be quite small.
2. It is better to make δs_{arb} small in the first place. This may save time in that the algorithm will not have to go through the whole cycle too many times before a successful convergence takes place. Also , it ensures that the first step is a small one and so the arc length between the first two points can be accurately approximated by assuming the curve is straight between those two points.

3. δs should really be scaled according to which component of $\mathbf{u}_{est}$ it is being added to. For example, adding 0.1 to an eccentricity of 0.001 changes the orbit a great deal more than adding 0.1 km to a semi-major axis of 8000 km.

The starter has parallels to the first five of the eight steps in the main cycle. It has step-size selection (arbitrary), it has a predictor (adding δs to components of $\mathbf{u}_{est}$), it has the Newton-Raphson iterative corrector, it has a monitor (if corrector does not converge try adding δs to a different component of $\mathbf{u}_{est}$) and it has an arc length corrector which calculates the arc length instead of simply using the δs that led to convergence.

The three of the eight steps in the main cycle that have no parallels in the starter are omitted for good reasons. The critical state collector needs three corrected points to make sense, and so cannot be included. It is justifiably assumed that we could not have followed the entire loop around in just one small step, and so a check for termination is not necessary at this stage. Since the algorithm has been set up such that the first step is a small one, the curve cannot cross the $\lambda = 1$ hyperplane after that first step. Thus, a solution state collector is not necessary at this stage.

3.2 Step Size Selection

The selection of the step size for the next step along the loop depends on the last step size and on the number of iterations the corrector required for convergence at the last step. The maximum number of iterations allowed is set to be 10. That is, if the corrector performs 10 iterations and has not converged to the limit required, then a non-convergence is recorded even though the corrector may have been converging slowly. The step size selector says that if the number of iterations was 9 or 10, then the next step size is equal to the last step size. If the number of iterations was less than 9, then the step size is chosen to be a factor of 1.4 times the last step size. My experience with the selection of step size indicates that it makes little difference what number of iterations is used as the limit between increasing step size and leaving it as is. However, Smith [1] suggests that a more complicated step size selection scheme

can lead to a more time efficient algorithm.

The effect of the step size selector is that the algorithm automatically uses relatively small steps in areas where the solution curve has a small radius of curvature, and uses relatively large steps in areas where the radius of curvature is large.

3.3 Predictor

The predictor of the next point on the loop uses Lagrange extrapolation based on the last three corrected points and the chosen step size to predict the next point, except when only two corrected points are known, i. e. , immediately after the starter. In that case, the extrapolation is based on only those two points. Note that the extrapolation could be based on any number of previous corrected points but the greater the number of points used, the more computation per step required, and there is no reason to believe the predictions will be better. Smith [1] notes that he found 3 or 4 to be an optimum number of back points to use. I found 2 or 3 to be optimum depending on the particular case.

The formulae used for the prediction are shown below, but first a little notation. To be consistent with notation already used, let s_1 be the arc length up to the last corrected point, s_2 be the arc length up to the one before last corrected point, etc. Let s_{pred} be the arc length at the predicted point, $s_{pred} = s_1 + \delta s$. Finally, let the number of back points being used be denoted by n. Set

$$\mathbf{u}_{pred} = \sum_{i=1}^{n} L_i \, \mathbf{u}_i \qquad i = 1, \ldots, n \tag{3.1}$$

where the Lagrange coefficients (L_i) are calculated according to :

$$L_i = \frac{\prod_{\substack{j=1 \\ j \neq i}}^{n} (s_{pred} - s_j)}{\prod_{\substack{j=1 \\ j \neq i}}^{n} (s_i - s_j)} \qquad i = 1, \ldots, n \tag{3.2}$$

3.4 Corrector

The algorithm arrives at the corrector with a guess at a point on the solution loop, $\mathbf{u}_{pred}$. The guess is corrected to a true point on the solution curve using an iterative Newton-Raphson scheme. We can take one of two approaches to the problem of correcting this guess. We can insist on keeping λ constant and correcting the $\mathbf{x}$ part of $\mathbf{u}_{pred}$. This approach has the potential for leading to problems. For example, if the curve is nearly parallel to a λ hyperplane, as it will be near critical points, then convergence may be slow, leading to prohibitively small steps in that area. This point is illustrated by figure 3-1.

The other approach, which is the one used in this thesis, is to allow λ to be a variable in the corrector scheme and to introduce the constraint that the correction at each iteration must be perpendicular to the tangent vector of the curve. For maximum accuracy the tangent vector should be calculated at the latest iterated value for the predicted point.

Each iteration involves solving a system of seven equations in the seven variables, λ, $\mathbf{x}$. One equation is derived from the correction vector constraint. The other six equations are derived from making a linear Taylor Series approximation of equation 2.4 which is reproduced here as equation 3.3

$$\mathbf{f}(\mathbf{x}) = \lambda \mathbf{y}_{ex} + (1 - \lambda)\mathbf{y}_{est} \tag{3.3}$$

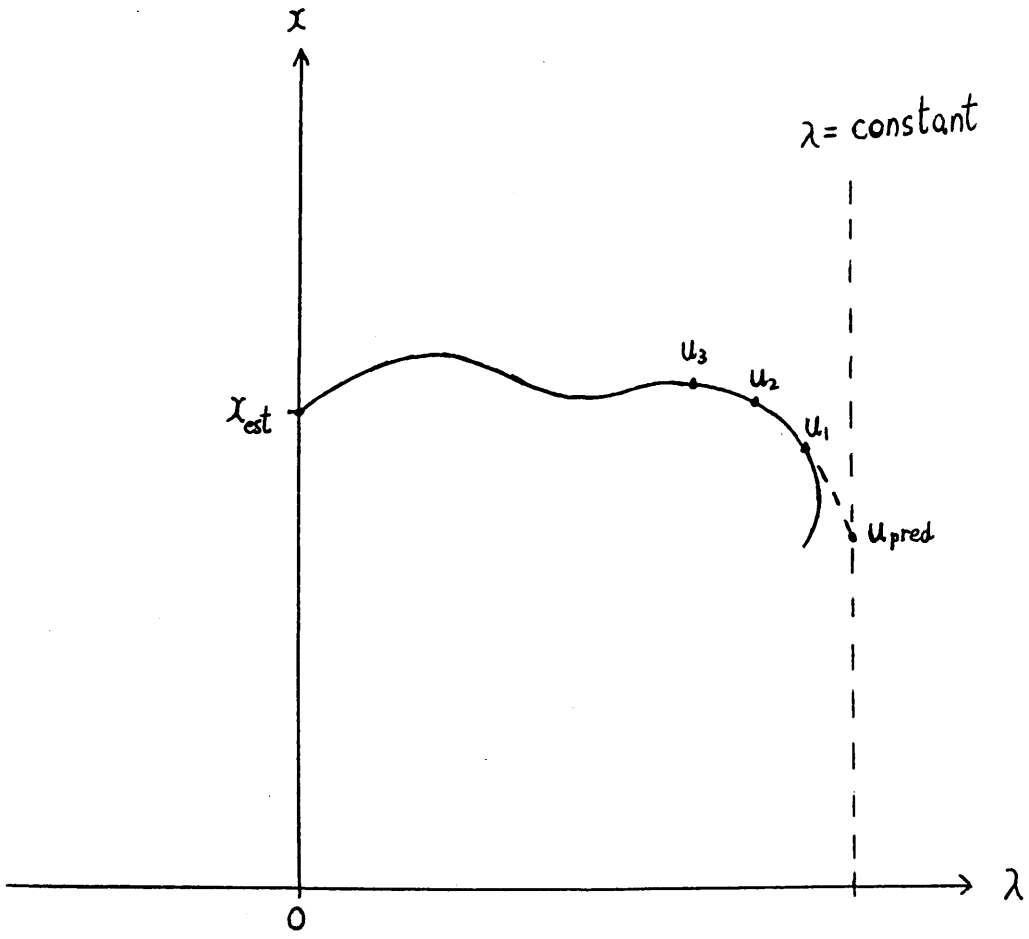

Figure 3-1: Schematic Diagram Illustrating How Correcting With Constant λ Could Cause Problems

The constraint equation is given by :

$$\frac{d\mathbf{u}}{ds} \cdot \delta\mathbf{u} = 0 \tag{3.4}$$

The evaluation of $\frac{d\mathbf{u}}{ds}$ is discussed in detail in the next chapter.

Let a point on the solution curve be denoted by $\mathbf{u}_{sol}$ where $\mathbf{u}_{sol} = (\lambda_{sol}, \mathbf{x}_{sol})$. This point will satisfy equation 3.3 giving :

$$\mathbf{f}(\mathbf{x}) \mid_{\mathbf{x}=\mathbf{x}_{sol}} = \lambda_{sol}\, \mathbf{y}_{ex} + (1 - \lambda_{sol})\, \mathbf{y}_{est} \tag{3.5}$$

Rewriting $\mathbf{u}_{sol}$ as :

$$\mathbf{u}_{sol} = \mathbf{u}_{pred} + \delta\mathbf{u} = (\lambda_{pred} + \delta\lambda, \mathbf{x}_{pred} + \delta\mathbf{x}) \tag{3.6}$$

and substituting equation 3.6 into equation 3.5 gives :

$$\mathbf{f}(\mathbf{x}) \mid_{\mathbf{x}=\mathbf{x}_{pred}+\delta\mathbf{x}} = (\lambda_{pred} + \delta\lambda)\, \mathbf{y}_{ex} + (1 - \lambda_{pred} - \delta\lambda)\, \mathbf{y}_{est} \tag{3.7}$$

Now expanding the left hand side of equation 3.7 in a Taylor series and discarding the non-linear terms gives :

$$\mathbf{f}(\mathbf{x}) \mid_{\mathbf{x}=\mathbf{x}_{pred}+\delta\mathbf{x}} \approx \mathbf{f}(\mathbf{x}) \mid_{\mathbf{x}=\mathbf{x}_{pred}} + \mathbf{f}'(\mathbf{x}) \mid_{\mathbf{x}=\mathbf{x}_{pred}} \delta\mathbf{x} \tag{3.8}$$

Equating the right hand sides of equation 3.7 and equation 3.8 gives :

$$\mathbf{f}(\mathbf{x}) \mid_{\mathbf{x}=\mathbf{x}_{pred}} + \mathbf{f}'(\mathbf{x}) \mid_{\mathbf{x}=\mathbf{x}_{pred}} \delta\mathbf{x} \approx (\lambda_{pred} + \delta\lambda)\, \mathbf{y}_{ex} + (1 - \lambda_{pred} - \delta\lambda)\, \mathbf{y}_{est} \tag{3.9}$$

Rearranging equation 3.9 gives us the six equations to go along with the constraint equation in making up the complete system of seven.

$$(\mathbf{y}_{ex} - \mathbf{y}_{est})\, \delta\lambda - \mathbf{f}'(\mathbf{x}) \mid_{\mathbf{x}=\mathbf{x}_{pred}} \delta\mathbf{x} \approx \lambda_{pred}(\mathbf{y}_{est} - \mathbf{y}_{ex}) - \mathbf{y}_{est} + \mathbf{f}(\mathbf{x}) \mid_{\mathbf{x}=\mathbf{x}_{pred}} \tag{3.10}$$

where $\mathbf{f(x)}\,|_{\mathbf{x}=\mathbf{x}_{pred}}$ is the set of six ranges calculated by propagating the predicted state to the six chosen times, and $\mathbf{f'(x)}\,|_{\mathbf{x}=\mathbf{x}_{pred}}$ is the six-by-six matrix of partial derivatives of those six ranges with respect to the six epoch orbital elements.

After the seven-by-seven system has been solved, the resulting $\delta\mathbf{u}$ is added to the predicted state to give the new predicted state to be used in the next iteration. This process is repeated until the convergence criterion has been met or until the maximum ten iterations have been completed.

3.5 Monitor

This step consists of checking to ensure that three conditions regarding the last corrected point are satisfied. First, the corrector must have converged to the stipulated criterion. Second, the difference in the curve tangent vector between the last two corrected points must be less than some stipulated amount. Last, the difference in the λ values for the last two corrected points must be less than some specified amount (0.1 was chosen in this case).

If any of the three conditions are not met then the last corrected point is discarded, the δs is halved, and the algorithm returns to the predictor step. If all of the three conditions are satisfied then the point is accepted and the backpoint information is updated.

The first condition is obviously required because if the corrector has not converged, then the point in **u** space it returns with is not on the solution curve. The convergence criterion will be discussed in the next chapter which deals with the corrector step in greater detail.

The second condition is included in order to prevent the algorithm from *doubling back* on itself. That is, when the algorithm comes to a part of the solution loop which is highly curved, it may turn around and follow the path along which it came, back to the start point, thereby allowing termination to occur without completion of the solution loop [see figure 3-2]. This condition also prevents the algorithm from skipping portions of the solution curve at points where the curve *pinches in* [see figure 3-3].

The condition is enforced by taking the dot product of the tangent vectors at the last two corrected points and requiring this dot product to be greater than 0.9. Smith [1] recommends that the dot product should exceed 0.95 in order to satisfy this condition, but I found that 0.90 is good enough and allows the algorithm to proceed faster since it is a looser criterion. The method used for evaluating the tangent vector at a point on the curve is discussed in the next chapter.

The third condition is not necessary but it is useful. Its purpose is to ensure that reasonably small steps in λ are taken so that when plots of the orbit elements vs λ are made, they provide an accurate picture [see figure 3-4]. This condition is not included in Smith's work [1] because he was concerned with maximizing efficiency, not with ensuring that the plots of the solution loops were accurate. I would say that the condition could be excluded when the algorithm is being used in a real situation.

3.6 Arc Length Correction

The purpose of this step is to evaluate s_1, the arc length along the solution curve up to the last corrected point, $\mathbf{u}_1$. The arc length s_2 up to the one before last corrected point, $\mathbf{u}_2$ is known, so we need only compute the arc length between the last two corrected steps and add that to s_2 to get s_1. Smith [1] says that the step size, δs, between $\mathbf{u}_2$ and $\mathbf{u}_{pred}$, may be a good enough estimate to the arc length between the last two corrected points to render this step unnecessary. That is, he feels that the change in arc length due to the correction of $\mathbf{u}_{pred}$ to $\mathbf{u}_1$ is negligible. I found cases in which the arc length can change significantly during the correction process, and so the arc length correction procedure is necessary.

An exact expression for the arc length between the last two corrected points, $\mathbf{u}_1$ and $\mathbf{u}_2$ is :

$$s_1 - s_2 = \int_{\mathbf{u}=\mathbf{u}_2}^{\mathbf{u}=\mathbf{u}_1} (d\mathbf{u} \cdot d\mathbf{u})^{1/2} = \int_{s=s_2}^{s=s_1} (\frac{d\mathbf{u}}{ds} \cdot \frac{d\mathbf{u}}{ds})^{1/2}\, ds \qquad (3.11)$$

Since s_1 is precisely the thing we are trying to evaluate, we need to approximate the s_1 in the upper limit of the integral in the above equation with s_{pred}. Having done that, we can use Gaussian quadrature to evaluate the integral. Appendix A contains

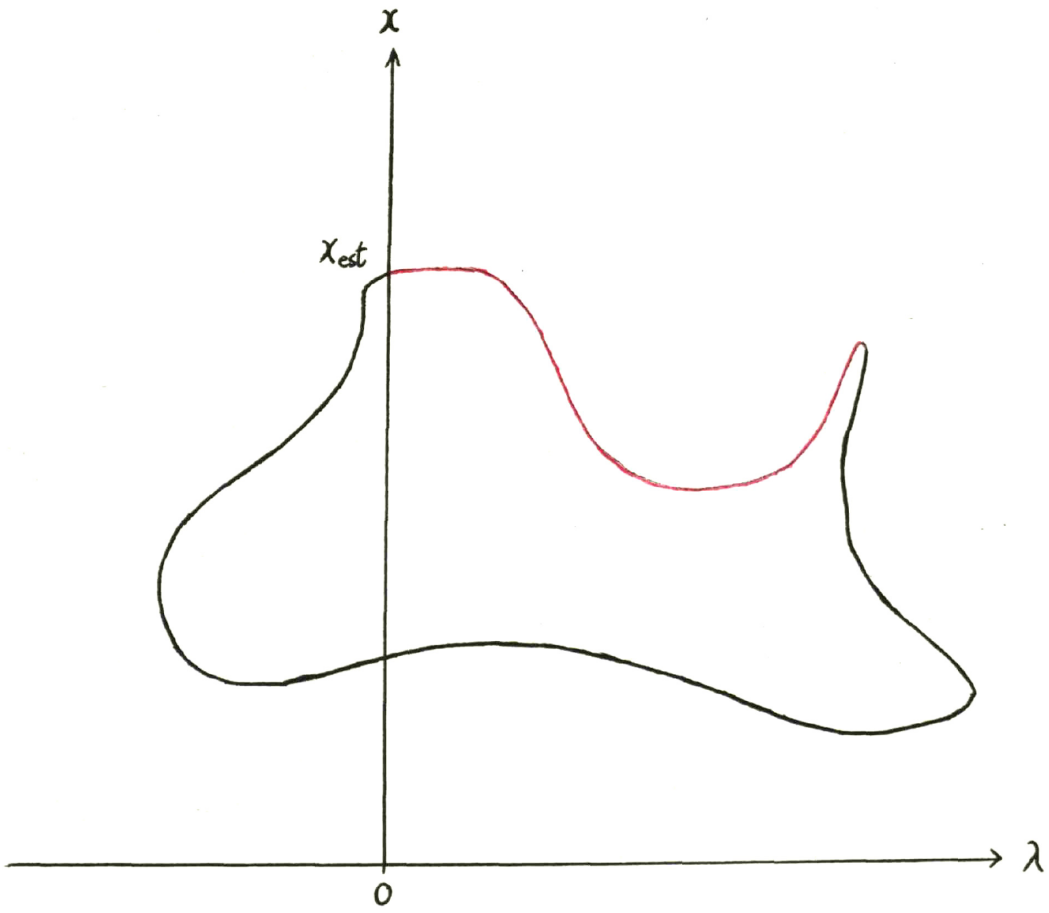

Figure 3-2: Schematic Example of the Solution Curve Doubling Back

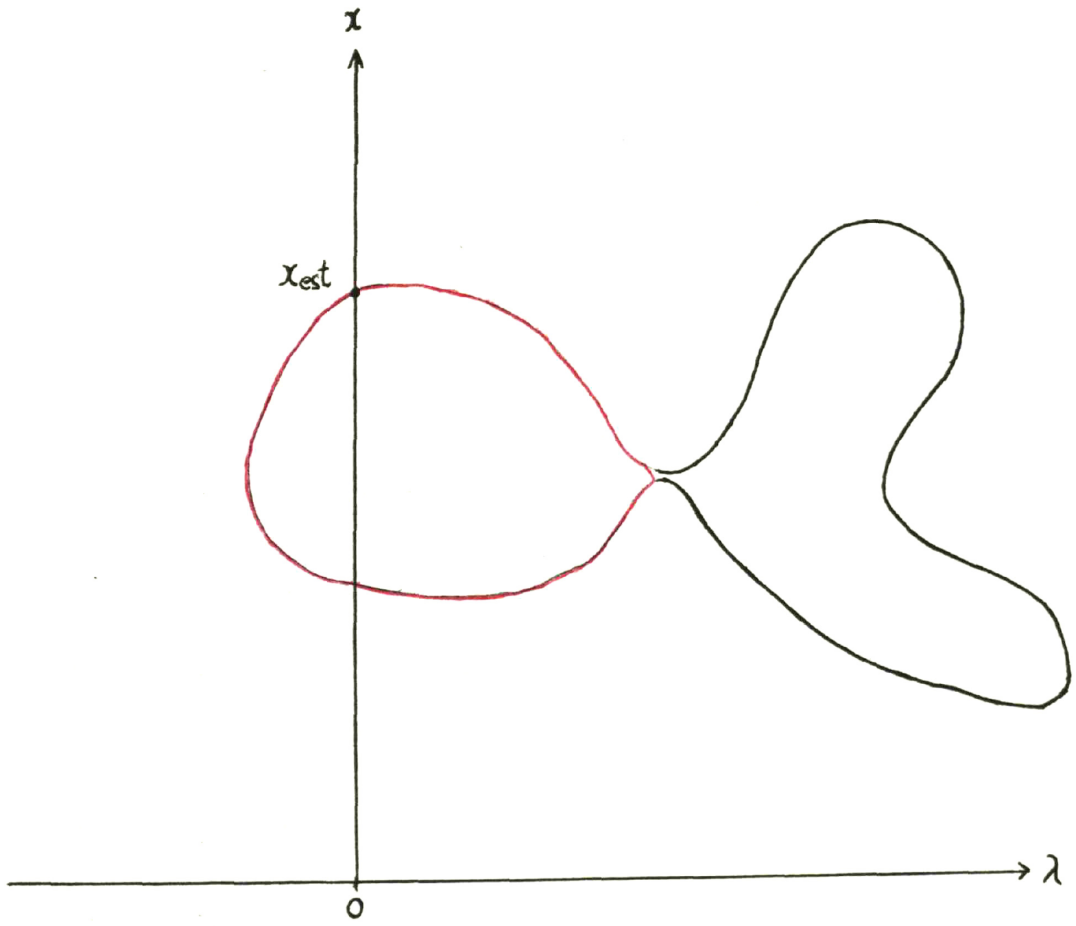

Figure 3-3: Schematic Example of the Solution Curve Skipping a Portion of the Solution Curve Where it Pinches In

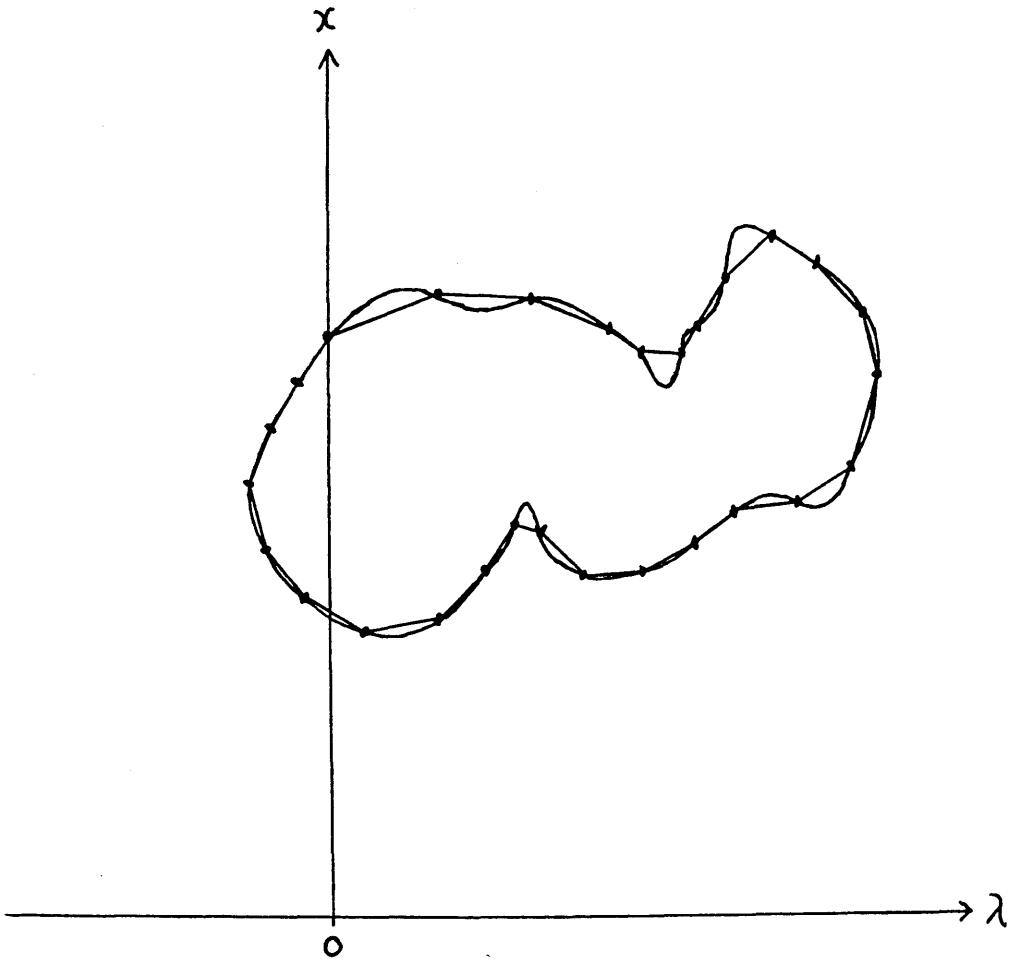

Figure 3-4: Schematic Diagram to Show the Effect of Limiting λ Step Between Consecutive Points on the Solution Curve

a description of the Gaussian quadrature method. Evaluation of the tangent vector $\frac{d\mathbf{u}}{ds}$ at the quadrature points, is done by differentiating the Lagrange interpolating polynomial discussed in the section on the predictor step. That produces :

$$\frac{d\mathbf{u}}{ds} = \sum_{i=1}^{n} \frac{dL_i}{ds}\,\mathbf{u_i} \qquad i = 1, \ldots, n \tag{3.12}$$

where

$$\frac{dL_i}{ds} = \frac{1}{\prod\limits_{\substack{j=1 \\ j \neq i}}^{n} (s_i - s_j)} \sum_{\substack{k=1 \\ k \neq i}}^{n} \left[\prod_{\substack{j=1 \\ j \neq i \\ j \neq k}}^{n} (s_{pred} - s_j) \right] \qquad i = 1, \ldots, n \tag{3.13}$$

Carrying out this quadrature should give us an improved estimate of the value of s_1. If we insert this improved value back into the upper limit of the integral, we can determine an even better estimate. The process can be iterated until changes in the estimate are less than some tolerance (1.0 e-8 was used in this case). This iteration represents a difference from Smith's work [1]: he only used one improvement of s_1.

Accurate recording of the arc length is useful for keeping the solution state collector, the critical state collector and the terminator sub-algorithms working efficiently.

3.7 Solution State Collection

This step detects when the solution loop has crossed the $\lambda = 1$ hyperplane, and then locates the exact point(s) on the solution curve which lie in this hyperplane. These points are stored in a file, so that after the algorithm has completed the loop, all of these solutions are available for the screening process.

Since each step can only have a maximum change of 0.1 in λ (see section on monitor), then if neither $\mathbf{u}_1$ nor $\mathbf{u}_2$ have a λ coordinate within 0.1 of 1.0, it is assumed the $\lambda = 1$ hyperplane has not been crossed and this step is skipped. Otherwise, the

following process is carried out.

1. Interpolated states are computed, using the Lagrange formulae shown in the Predictor section, at equal intervals of arc length between s_1 and s_2. In this case, the arc length is divided into 50 equal intervals requiring 49 interpolated states to be computed. The notation used for these interpolated states is that the n^{th} interpolated state starting from the $\mathbf{u}_2$ side, is denoted by $\mathbf{u}_n^{int}$.

2. If the sign of $(1\text{-}\lambda_{n+1}^{int})$ is opposite to the sign of $(1\text{-}\lambda_n^{int})$, the two interpolated states, $\mathbf{u}_{n+1}^{int}$ and $\mathbf{u}_n^{int}$, are corrected using the corrector sub-algorithm. The corrected states are denoted by $\mathbf{u}_{up}$ and $\mathbf{u}_{down}$ respectively.

3. A linearly interpolated state, $\mathbf{u}_{app}$, at $\lambda = 1$ is computed according to :

$$\mathbf{u}_{app} = \mathbf{u}_{down} + \frac{(1 - \lambda_{down})}{(\lambda_{up} - \lambda_{down})}(\mathbf{u}_{up} - \mathbf{u}_{down}) \tag{3.14}$$

4. This $\mathbf{u}_{app}$ is corrected using the corrector sub-algorithm

5. If the corrected λ_{app} is equal to 1 within a specified tolerance (1.0 e-8 in this case), then this $\mathbf{u}_{app}$ is stored as a solution. If not, whichever of $\mathbf{u}_{up}$ or $\mathbf{u}_{down}$ has a λ component farther from 1.0, is replaced by $\mathbf{u}_{app}$. A new $\mathbf{u}_{app}$ is computed as before, and this process is repeated until the corrected λ_{app} is equal to 1.0 within the specified tolerance.

There are two conceivable situations that could cause problems with this scheme. One possibility is that the interpolated curve between $\mathbf{u}_1$ and $\mathbf{u}_2$ crosses the $\lambda = 1$ hyperplane, but the solution curve does not. For example, see figure 3-5. This situation would lead to a failure to converge in the corrector sub-algorithm. If this happens, the last curve point, $\mathbf{u}_1$, is discarded, δs is halved and and a new $\mathbf{u}_1$ is computed.

The other potential problem occurs if the solution curve crosses the $\lambda = 1$ hyperplane, but the interpolated curve does not. The result of this would be that the algorithm would continue without having recorded a solution point where a solution

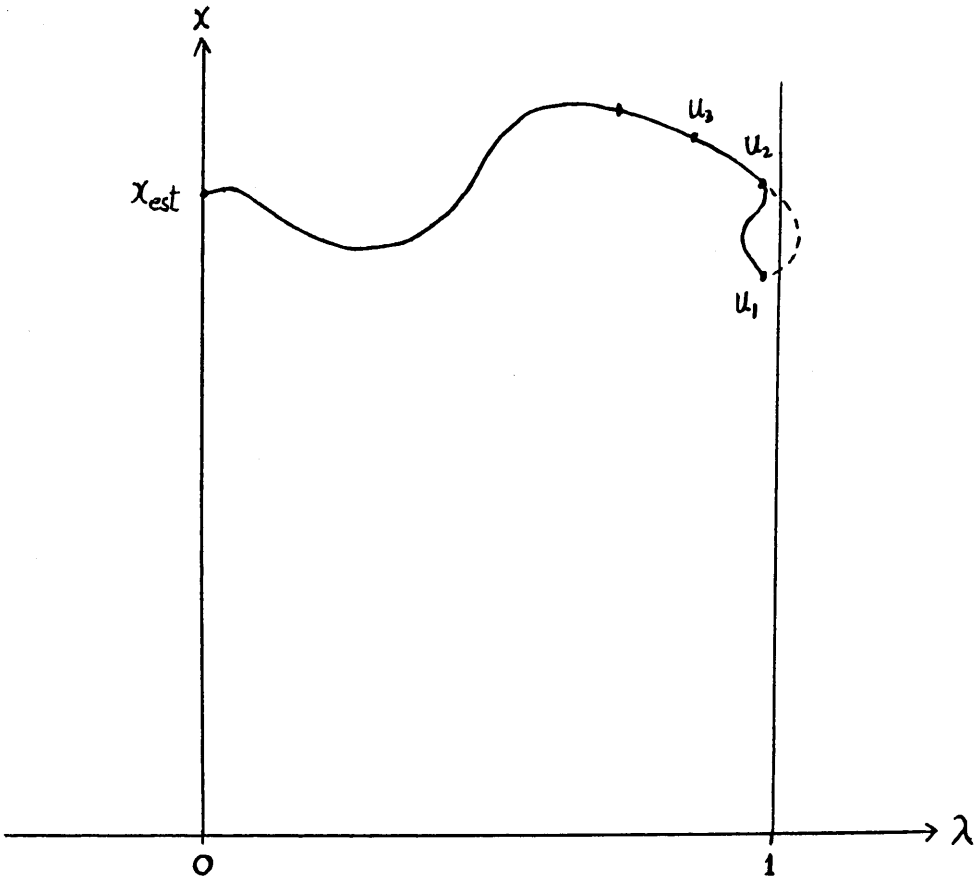

Figure 3-5: Schematic Diagram Showing Potential Problem With Solution State Collector

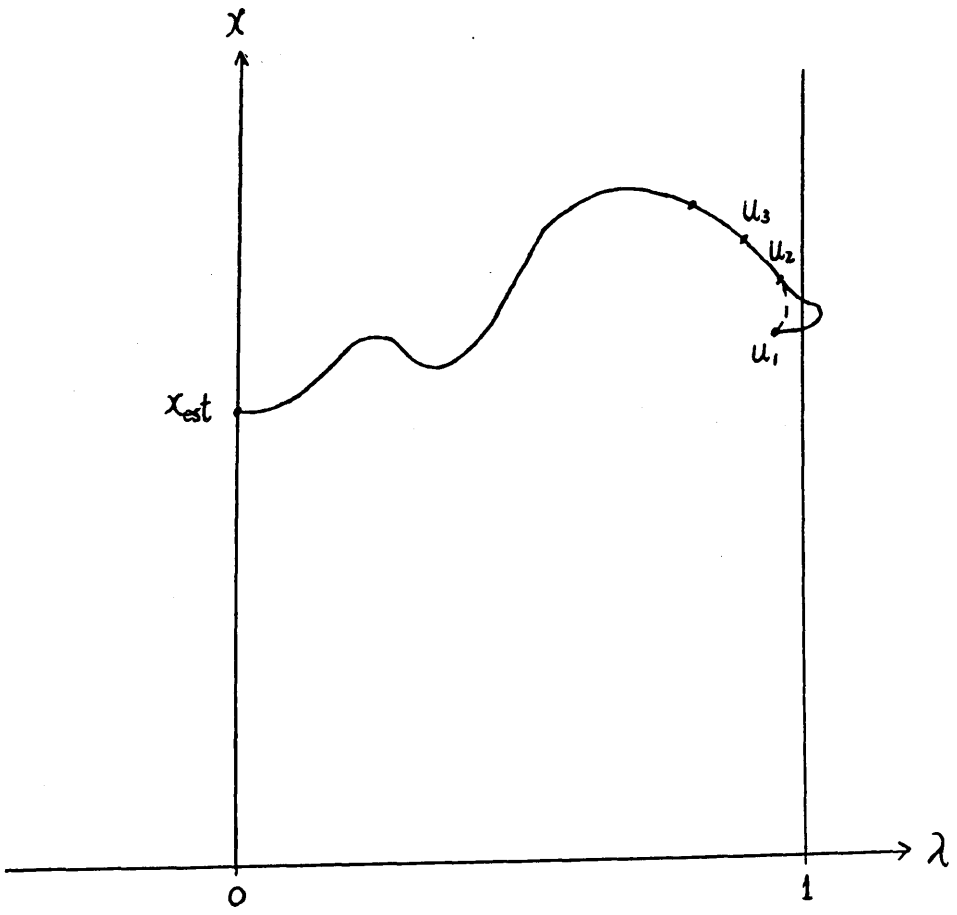

Figure 3-6: Schematic Diagram Showing Another Potential Problem With Solution State Collector

point exists. See figure 3-6. Actually, *two* exact solutions will fail to be recorded. Thankfully, this situation must be extremely rare. If the user checks the critical points (see next section) and finds that there is a critical point with λ equal to a tiny bit more or less than 1, then he/she should check the possibility that this situation may have occurred.

3.8 Critical State Collection

As mentioned in the previous chapter, the critical states are those points on the solution curve which are local extrema in λ. The reason for wanting to determine and record such states is related to the possibility of finding that the solution loop on which your a priori estimate lies, and the solution loop on which the solution which we seek lies, are not the same. There is an extended version of this homotopy continuation method that is entirely global. That is, it can follow all existing loops, not just the one on which the a priori estimate lies. Time did not permit the inclusion of that extension in this thesis, but some detailed description of the extension is included in Smith's paper [1].

A critical state collector was included in this thesis because initially, it was hoped that there would have been time to tackle the extended method. Also, if anyone should ever want to add the extension to this piece of work, they would have available a working critical point collector.

The first step of this sub-algorithm is to check, for λ_1, λ_2 and λ_3 corresponding to the last three points on the solution curve, whether either of the following two conditions exist.

$$\lambda_2 < \lambda_1 \text{ and } \lambda_2 < \lambda_3$$

$$\lambda_2 > \lambda_1 \text{ and } \lambda_2 > \lambda_3$$

If neither of these two conditions exists, it is assumed that there are no local extrema in λ between $\mathbf{u}_1$ and $\mathbf{u}_3$. If the first condition exists, it is assumed that there is a local minimum in λ between $\mathbf{u}_1$ and $\mathbf{u}_3$. If the second condition exists, it is assumed

that there is a local maximum in λ between $\mathbf{u}_1$ and $\mathbf{u}_3$.

If it has been determined that a critical point does exist, the task of locating it is carried out. It should be pointed out that the location technique used in this thesis is quite different from the one suggested by Smith [1].

The location process starts by calculating the arc length in the range $[s_1, s_3]$ for which the derivative of λ with respect to arc length, on the quadratic Lagrange interpolated curve, is equal to zero. This value of arc length is denoted by s_{extrm}. It is assumed that the critical point lies within the restricted range defined by $s_{extrm} \pm 0.2(s_1 - s_3)$. The s value corresponding to the +ve sign is denoted by s_{high}, and the s value corresponding to the -ve sign is denoted by s_{low}. The purpose of this first step is to reduce the range in which the critical point lies to 40% of the original range. The Lagrange interpolation formulae are used to find approximate points on the solution curve at s_{high}, s_{low} and s_{extrm}. These three approximate points are then corrected using the corrector sub-algorithm to give the points on the solution curve denoted by $\mathbf{u}_{low}, \mathbf{u}_{extrm}$ and $\mathbf{u}_{high}$. The arc lengths associated with these points are then corrected using the arc length corrector sub-algorithm. From this point the location technique consists of a series of reductions of the range of arc lengths, defined by $[s_{high}, s_{low}]$, within which the critical point lies. When this range is reduced to the point where $s_{high} - s_{low} < 0.01$, and where $\mid \lambda_{high} - \lambda_{low} \mid < 1.0$ e-7, then the critical point is equated to $\mathbf{u}_{extrm}$, a point between $\mathbf{u}_{high}$ and $\mathbf{u}_{low}$.

Each range reduction step involves the following :

- Compute an arc length half-way between s_{low} and s_{high} and call that arc length s_{new}.
- Use the Lagrange interpolator to approximate a point on the solution curve at s_{new}.
- Correct that point using the corrector sub-algorithm to a point on the solution curve, $\mathbf{u}_{new}$.
- Correct the arc length s_{new} using the arc length corrector sub-algorithm.

- According to the relative values of $\lambda_{high}, \lambda_{low}, \lambda_{new}$ and λ_{extrm}, discard either $\mathbf{u}_{high}$ or $\mathbf{u}_{low}$. Reset the other three to $\mathbf{u}_{high}, \mathbf{u}_{low}$ and $\mathbf{u}_{extrm}$ and repeat the process until the criteria on arc length and λ are satisfied.

If the critical point is a maximum, four scenarios for discarding a point and resetting the other three exist. They are outlined in figure 3-7. In the first scenario shown the process is :

$$\mathbf{u}_{new} \rightarrow \mathbf{u}_{high}$$
$$\mathbf{u}_{extrm} \rightarrow \mathbf{u}_{extrm}$$
$$\mathbf{u}_{low} \rightarrow \mathbf{u}_{low}$$
$$\mathbf{u}_{high} \rightarrow \textit{discarded}$$

In the second it is :

$$\mathbf{u}_{new} \rightarrow \mathbf{u}_{low}$$
$$\mathbf{u}_{extrm} \rightarrow \mathbf{u}_{extrm}$$
$$\mathbf{u}_{high} \rightarrow \mathbf{u}_{high}$$
$$\mathbf{u}_{low} \rightarrow \textit{discarded}$$

In the third it is :

$$\mathbf{u}_{new} \rightarrow \mathbf{u}_{extrm}$$
$$\mathbf{u}_{extrm} \rightarrow \mathbf{u}_{low}$$
$$\mathbf{u}_{high} \rightarrow \mathbf{u}_{high}$$
$$\mathbf{u}_{low} \rightarrow \textit{discarded}$$

In the fourth it is :

$$\mathbf{u}_{new} \rightarrow \mathbf{u}_{extrm}$$
$$\mathbf{u}_{extrm} \rightarrow \mathbf{u}_{high}$$
$$\mathbf{u}_{low} \rightarrow \mathbf{u}_{low}$$
$$\mathbf{u}_{high} \rightarrow \textit{discarded}$$

Parallel situations exist if the critical point is a minimum.

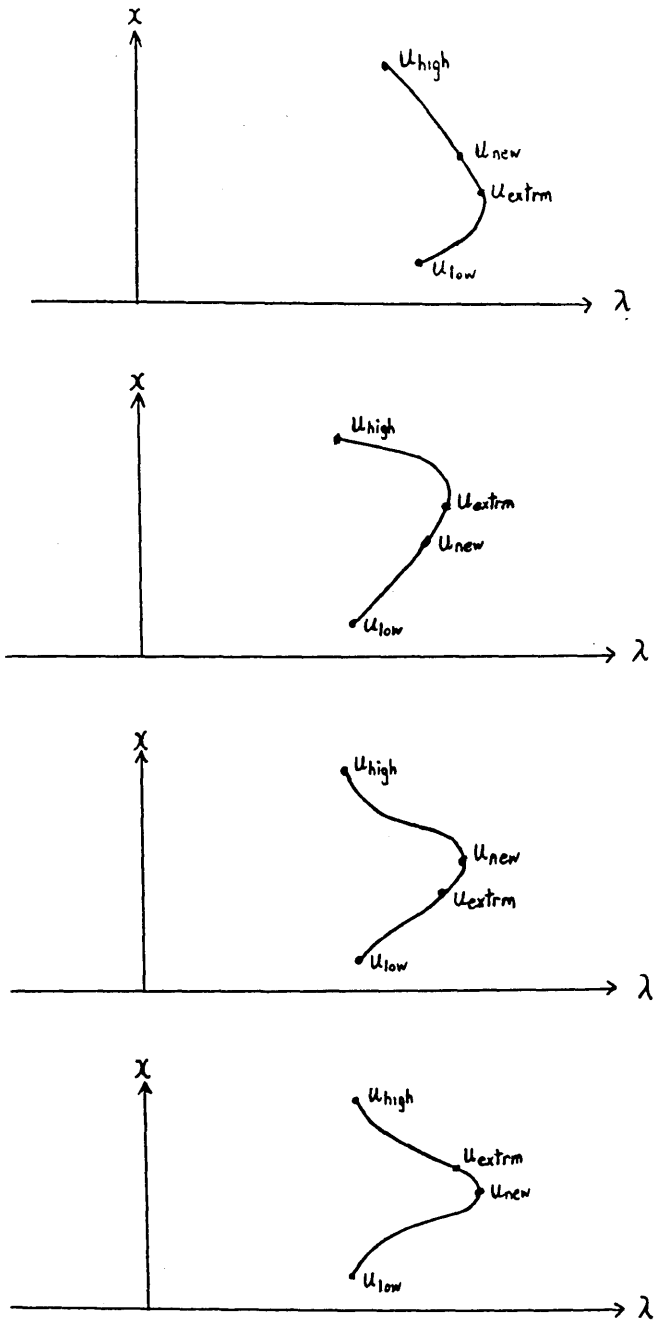

Figure 3-7: Schematic Diagrams Representing the Different Discard and Reset Scenarios

NOTE : When a critical point has been determined it is compared with the critical points that have already been recorded. If it is found to be the same as one of the already recorded critical points, that message is recorded instead of recording the same critical point again. The same is true of collection of the solution states. This kind of occurrence indicates either a doubling back of the algorithm, or a situation in which the terminator has failed to recognize a return to the initial start point causing the solution loop to be followed twice. Neither of these things should happen, but at least if they do, there will be evidence to show that they have.

3.9 Termination of the Algorithm

Termination of the algorithm may occur in one of three different ways. The solution loop can be completed, i. e. , the algorithm followed the loop from the initial point, all the way around and back to the initial point. This will be called normal termination. Prior to the completion of the loop, the step size selected may become less than some specified tolerance. If that happens, the user will be offered the option of terminating. This is an example of optional termination. Prior to the completion of the loop, termination may occur, for example, due to the number of solution points exceeding some maximum allowed number. This is referred to as abnormal termination.

3.9.1 Normal Termination

The normal termination sub-algorithm is very similar to the solution state collector scheme. First it is determined whether the loop has crossed the $\lambda = 0$ hyperplane (in the solution state collector the $\lambda = 1$ hyperplane was the one of interest). If it has, the exact points at which that occurs are determined in an identical manner to that used in the solution state collector. Now, instead of storing the $\lambda = 0$ states, this sub-algorithm compares them with the initial starting point. If a $\lambda = 0$ state is found to be identical to the initial starting point to within some tolerance, then it is clear that the loop has been completed, and the algorithm undergoes normal termination.

3.9.2 Optional Termination

There are two circumstances under which termination is offered as an option to the user. The one which is more likely to occur is that the step size becomes less than some tolerance (1.0 e-5 in this case). This can happen if, for example, the algorithm reaches a part of the solution loop which has hyperbolic orbit states on it. Since the orbit propagation subroutines used in this thesis are only designed to be able to handle elliptic orbits, this will cause the corrector to fail, and therefore the step size to be halved, until the step size is less than the tolerance value. The less likely possibility that leads to optional termination is that the number of stored critical points becomes equal to some specified maximum (10 in this case).

In either case, the options that are offered are :

1. terminate algorithm

2. restart algorithm at the original start point

3. restart algorithm at the last corrected point on the solution curve

If step size is the reason for the options being offered, restarting at the last corrected point is not recommended because either the algorithm just runs right back into the same problem that caused the step size to be so small in the first place, or it heads off along the path that it took to get there. That is, the restart causes a doubling back of the algorithm. Restarting at the initial start point may be useful in some cases.

If the number of critical points is the reason for the options being offered, any of the options may prove useful.

3.9.3 Abnormal Termination

One situation that causes the algorithm to terminate prior to completing the solution loop without option, is that the number of solution points recorded exceeds a set maximum number (10 in this case). Other situations which can lead to this kind of termination are :

- Program crashes. Hopefully, this never happens !!

- Starter step fails either at the very beginning or after a restart option has been chosen. If it happens at the very beginning it is indicative of a probable bug in the software. Otherwise, it is probably due to the situation mentioned in the section on optional termination. That is, where even the tiniest step leads to a point that represents a hyperbolic orbit.

- Normally, if the corrector sub-algorithm fails when called from the solution state collector sub-algorithm or from the normal terminator sub-algorithm, the last corrected point is discarded, the step size is halved, and the algorithm continues. However, if this happens just after having restarted, the algorithm terminates.

Chapter 4

Details of the Newton-Raphson Corrector Scheme

The major computational step in the method is the corrector step. This chapter describes that step in detail.

As was indicated in the previous chapter, the corrector sub-algorithm takes a predicted state and corrects it to a state on the solution curve using a Newton-Raphson iterative scheme. Each iteration involves solving a seven-by-seven linear system to get a $\delta \mathbf{u}$ correction to the latest prediction. That system is represented by equation 3.4 and equation 3.10 which are reproduced here for convenience.

$$\frac{d\mathbf{u}}{ds} \cdot \delta \mathbf{u} = 0$$

$$(\mathbf{y}_{ex} - \mathbf{y}_{est})\,\delta\lambda - \mathbf{f}'(\mathbf{x})\,|_{\mathbf{x}=\mathbf{x}_{pred}}\,\delta\mathbf{x} \approx \lambda_{pred}(\mathbf{y}_{est} - \mathbf{y}_{ex}) - \mathbf{y}_{est} + \mathbf{f}(\mathbf{x})\,|_{\mathbf{x}=\mathbf{x}_{pred}}$$

Since the $\mathbf{y}_{est}$ and $\mathbf{y}_{ex}$ vectors are constants which are specified near the beginning of the algorithm, the only quantities that remain to be calculated at each iteration are :

1. The set of ranges $\mathbf{f}(\mathbf{x})\,|_{\mathbf{x}=\mathbf{x}_{pred}}$ computed by propagating the last predicted state. This will suitably be denoted by $\mathbf{y}_{pred}$.

2. The six-by-six matrix of partial derivatives of ranges with respect to the six

epoch orbital elements $\mathbf{f}'(\mathbf{x})\,|_{\mathbf{x}=\mathbf{x}_{pred}}$. This will suitably be denoted by $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$. The $_{pred}$ subscript has been dropped from the $\mathbf{y}$ and the $\mathbf{x}$ for clarity, although the reader should keep in mind that strictly speaking they should be there.

3. The tangent vector $\frac{d\mathbf{u}}{ds}$.

The range at a given time t after epoch is calculated from the epoch elements by first propagating the epoch elements, $\mathbf{x}$, to that time to get the current elements, $\mathbf{x}_t$, using whichever dynamics model was chosen (this thesis employs either two-body mechanics or Brouwer-Lyddane), then by converting those current elements to a satellite position in inertial Cartesian coordinates. Finally, knowing the station position in inertial Cartesian coordinates it is simple to calculate the range between station and satellite. Thus, carrying out this process using the last predicted state as epoch, and using the six measurement times, we arrive at the required ranges $\mathbf{y}_{pred}$.

The calculation of the matrix of partial derivatives $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ is more complicated. One method is to evaluate the partial derivatives analytically. The technique used is to calculate them one row at a time. The n^{th} row of the matrix, denoted by $\frac{\partial y^{(n)}}{\partial \mathbf{x}}$, where

$$\mathbf{y} = (y^{(1)}, y^{(2)}, y^{(3)}, y^{(4)}, y^{(5)}, y^{(6)})$$

is calculated from the product of three matrices as follows :

$$\frac{\partial y^{(n)}}{\partial \mathbf{x}} = \frac{\partial y^{(n)}}{\partial \mathbf{pv}} \cdot \frac{\partial \mathbf{pv}}{\partial \mathbf{x}_t} \cdot \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}} \tag{4.1}$$

The $\mathbf{pv}$ denotes the vector of current position and velocity in Cartesian coordinates. Thus, $\frac{\partial \mathbf{pv}}{\partial \mathbf{x}_t}$ represents the six-by-six matrix of the partial derivatives of current position and velocity with respect to current elements. A subroutine called CRTDRV [7], which calculates this matrix, provided that equinoctial elements are being used, was already available.

The row vector, $\frac{\partial y^{(n)}}{\partial \mathbf{pv}}$, which represents the partial differentiation of the n^{th} range with respect to current position and velocity, is relatively simple. Let $\mathbf{pv}$ be expressed

as :

$$\mathbf{pv} = (p_x, p_y, p_z, v_x, v_y, v_z)$$

Also, let the current station position be represented by :

$$(r_x, r_y, r_z)$$

Then we can express $y^{(n)}$ as :

$$y^{(n)} = \sqrt{(p_x - r_x)^2 + (p_y - r_y)^2 + (p_z - r_z)^2}$$

Simple differentiation thus leads to :

$$\frac{\partial y^{(n)}}{\partial \mathbf{pv}} = \left(\frac{p_x - r_x}{y^{(n)}}, \frac{p_y - r_y}{y^{(n)}}, \frac{p_z - r_z}{y^{(n)}}, 0, 0, 0 \right)$$

The relationship between current and epoch equinoctial elements, under the two-body mechanics system, is simply given by :

$$a_t = a$$
$$h_t = h$$
$$k_t = k$$
$$p_t = p$$
$$q_t = q$$
$$m_t = m + t\sqrt{\frac{\mu}{a^3}}$$

where μ is the gravitational parameter of the Earth and t is the time elapsed since epoch. Therefore there are only seven non-zero entries in the matrix $\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}}$ — six $1^{'s}$ on the diagonal and an entry in the bottom left corner given by $-\frac{3}{2}t\sqrt{\frac{\mu}{a^5}}$. The value of

t will depend on which row is being computed.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ -\frac{3}{2}t\sqrt{\frac{\mu}{a^5}} & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Therefore, analytical evaluation of $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ is a good technique for two-body mechanics propagation.

If instead Brouwer-Lyddane propagation is being used, then the relationship between current and epoch equinoctial elements is much more complicated than for two-body mechanics. We can take one of three approaches with Brouwer-Lyddane. We can pretend that the effect of Brouwer-Lyddane on the partial derivatives is negligible, since the only difference between Brouwer-Lyddane and two-body mechanics is that Brouwer-Lyddane includes the first four harmonic terms in the gravity field expansion. Taking this approach we can leave the calculation of partial derivatives in the same form as it was for two-body mechanics. This is the approach suggested by Smith [1]. The second approach is to try to calculate the partial derivatives analytically from the Brouwer theory. This is intellectually more satisfactory but a great deal of work for possibly little gain. The third approach, which is more of an engineering compromise approach, is to compute the partial derivatives in $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ by numerical differencing.

The second approach was ruled out because of the amount of time it would have taken to develop a subroutine that calculates analytical partial derivatives for the Brouwer-Lyddane propagation model. Both the first and the third approaches were tried and the results compared. The first approach requires substantially less computation per step but gives less accurate partial derivatives than the third approach. The effect of the less accurate partial derivatives is generally to cause the algorithm to require smaller step sizes along the curve and consequently to require more steps

to complete the loop. In fact, I found that in areas where the curve had a small radius of curvature the step size was often reduced to a value smaller than the minimum allowed and so the algorithm terminated without completing the loop, whereas using numerically determined partials allowed completion of the loop. Based on this finding, I chose to implement the numerical differencing method of calculating partial derivatives for the Brouwer-Lyddane propagation model.

The method of evaluating $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ by numerical differencing is also a row by row process. We start with the set of six ranges

$$\mathbf{y} = (y^{(1)}, y^{(2)}, y^{(3)}, y^{(4)}, y^{(5)}, y^{(6)})$$

which have been computed by propagating the predicted epoch state

$$\mathbf{x} = (x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}, x^{(5)}, x^{(6)})$$

(remember the $_{pred}$ subscript has been dropped from both $\mathbf{y}$ and $\mathbf{x}$), to the six observation times

$$\mathbf{t} = (t^{(1)}, t^{(2)}, t^{(3)}, t^{(4)}, t^{(5)}, t^{(6)})$$

To compute $\frac{\partial y^{(n)}}{\partial x^{(m)}}$, the m^{th} entry of the n^{th} row of the partial derivative matrix $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$, propagate the epoch element vector formed by adding $\delta x^{(m)}$ to $x^{(m)}$ entry in $\mathbf{x}$, to the time $t^{(n)}$. Then calculate the resulting range $y^{(res)}$. The desired entry is given by :

$$\frac{\partial y^{(n)}}{\partial x^{(m)}} \approx \frac{y^{(res)} - y^{(n)}}{\delta x^{(m)}}$$

$\delta x^{(m)}$ should be some very small value and should also be scaled according to which component of $\mathbf{x}$ is being dealt with for the same reason that the δs in the starter scheme needed to be scaled. In this thesis $\delta x^{(m)}$ is equal to 1.0 e-8 times the relevant scaling factor.

Finally, we need to address the calculation of the tangent vector, $\frac{d\mathbf{u}}{ds}$. One way of calculating the tangent vector is to use the derivatives of the Lagrange interpolating formulae of equations 3.1 and 3.2. This method is discussed in the previous chapter in

the section on the arc length corrector sub-algorithm, and the relevant equations are equation 3.12 and equation 3.13. This is the method used by Smith [1], but he does recommend a second and better method. Initially, in this thesis, the first method was implemented. Later on however, the switch was made to the second method, and it was indeed found to perform better. That second method is outlined as follows.

Differentiation of equation 3.3 with respect to arc length s leads to :

$$(\mathbf{y}_{ex} - \mathbf{y}_{est})\frac{d\lambda}{ds} - \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \cdot \frac{d\mathbf{x}}{ds} = 0 \tag{4.2}$$

Also, in the limit, $\delta\mathbf{u} \cdot \delta\mathbf{u} = (\delta s)^2$, which implies :

$$\left(\frac{d\lambda}{ds}\right)^2 + \left(\frac{d\mathbf{x}}{ds} \cdot \frac{d\mathbf{x}}{ds}\right) = 1 \tag{4.3}$$

If we let the 6-by-1 column vector, $(\mathbf{y}_{ex} - \mathbf{y}_{est})$, be represented by $\mathbf{c}$, and we let the 6-by-6 Jacobian matrix, $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$, be represented by $\mathbf{B}$, then equation 4.2 may be written as :

$$\mathbf{c}\frac{d\lambda}{ds} - \mathbf{B}\frac{d\mathbf{x}}{ds} = 0$$

$$\Rightarrow \qquad \frac{d\mathbf{x}}{ds} = \mathbf{B}^{-1}\mathbf{c}\frac{d\lambda}{ds} \tag{4.4}$$

Substituting equation 4.4 into equation 4.3 gives :

$$\left(\frac{d\lambda}{ds}\right)^2 \left[1 + (\mathbf{B}^{-1}\mathbf{c}) \cdot (\mathbf{B}^{-1}\mathbf{c})\right] = 1$$

$$\Rightarrow \qquad \left(\frac{d\lambda}{ds}\right)^2 \left[1 + (\mathbf{B}^{-1}\mathbf{c})^T(\mathbf{B}^{-1}\mathbf{c})\right] = 1$$

Thus, letting D be defined by :

$$D = [1 + (\mathbf{B}^{-1}\mathbf{c})^T(\mathbf{B}^{-1}\mathbf{c})]^{\frac{1}{2}} = [1 + \mathbf{c}^T\mathbf{B}^{-T}\mathbf{B}^{-1}\mathbf{c}]^{\frac{1}{2}}$$

we can write :

$$\frac{d\mathbf{u}}{ds} = \left(\frac{d\lambda}{ds}, \frac{d\mathbf{x}}{ds}\right) = \left(\frac{1}{D}, \frac{\mathbf{B}^{-1}\mathbf{c}}{D}\right) \tag{4.5}$$

This is the expression used in computing the tangent vector at each iteration. Note that **B** and **c** have already been computed at this stage so that this tangent evaluation method involves only one non-simple step, i. e. , inverting the **B** matrix.

In summary, each iteration in the corrector sub-algorithm consists of :

1. Solving the linear seven-by-seven system for

$$\delta \mathbf{u} = \begin{pmatrix} \delta\lambda \\ \delta x^{(1)} \\ \delta x^{(2)} \\ \delta x^{(3)} \\ \delta x^{(4)} \\ \delta x^{(5)} \\ \delta x^{(6)} \end{pmatrix}$$

2. Adding $\delta \mathbf{u}$ to the previous **u** approximation to get the updated approximation

3. Testing the updated approximation against the convergence criterion

The system is written explicitly as :

$$\mathbf{A} \begin{pmatrix} \delta\lambda \\ \delta x^{(1)} \\ \delta x^{(2)} \\ \delta x^{(3)} \\ \delta x^{(4)} \\ \delta x^{(5)} \\ \delta x^{(6)} \end{pmatrix} = \begin{pmatrix} 0 \\ \lambda[y_{est}^{(1)} - y_{ex}^{(1)}] - y_{est}^{(1)} + y^{(1)} \\ \lambda[y_{est}^{(2)} - y_{ex}^{(2)}] - y_{est}^{(2)} + y^{(2)} \\ \lambda[y_{est}^{(3)} - y_{ex}^{(3)}] - y_{est}^{(3)} + y^{(3)} \\ \lambda[y_{est}^{(4)} - y_{ex}^{(4)}] - y_{est}^{(4)} + y^{(4)} \\ \lambda[y_{est}^{(5)} - y_{ex}^{(5)}] - y_{est}^{(5)} + y^{(5)} \\ \lambda[y_{est}^{(6)} - y_{ex}^{(6)}] - y_{est}^{(6)} + y^{(6)} \end{pmatrix}$$

where

$$\mathbf{A} = \begin{pmatrix} \frac{\partial \lambda}{\partial s} & \frac{\partial x^{(1)}}{\partial s} & \frac{\partial x^{(2)}}{\partial s} & \frac{\partial x^{(3)}}{\partial s} & \frac{\partial x^{(4)}}{\partial s} & \frac{\partial x^{(5)}}{\partial s} & \frac{\partial x^{(6)}}{\partial s} \\ (y_{ex}^{(1)} - y_{est}^{(1)}) & -\frac{\partial y^{(1)}}{\partial x^{(1)}} & -\frac{\partial y^{(1)}}{\partial x^{(2)}} & -\frac{\partial y^{(1)}}{\partial x^{(3)}} & -\frac{\partial y^{(1)}}{\partial x^{(4)}} & -\frac{\partial y^{(1)}}{\partial x^{(5)}} & -\frac{\partial y^{(1)}}{\partial x^{(6)}} \\ (y_{ex}^{(2)} - y_{est}^{(2)}) & -\frac{\partial y^{(2)}}{\partial x^{(1)}} & -\frac{\partial y^{(2)}}{\partial x^{(2)}} & -\frac{\partial y^{(2)}}{\partial x^{(3)}} & -\frac{\partial y^{(2)}}{\partial x^{(4)}} & -\frac{\partial y^{(2)}}{\partial x^{(5)}} & -\frac{\partial y^{(2)}}{\partial x^{(6)}} \\ (y_{ex}^{(3)} - y_{est}^{(3)}) & -\frac{\partial y^{(3)}}{\partial x^{(1)}} & -\frac{\partial y^{(3)}}{\partial x^{(2)}} & -\frac{\partial y^{(3)}}{\partial x^{(3)}} & -\frac{\partial y^{(3)}}{\partial x^{(4)}} & -\frac{\partial y^{(3)}}{\partial x^{(5)}} & -\frac{\partial y^{(3)}}{\partial x^{(6)}} \\ (y_{ex}^{(4)} - y_{est}^{(4)}) & -\frac{\partial y^{(4)}}{\partial x^{(1)}} & -\frac{\partial y^{(4)}}{\partial x^{(2)}} & -\frac{\partial y^{(4)}}{\partial x^{(3)}} & -\frac{\partial y^{(4)}}{\partial x^{(4)}} & -\frac{\partial y^{(4)}}{\partial x^{(5)}} & -\frac{\partial y^{(4)}}{\partial x^{(6)}} \\ (y_{ex}^{(5)} - y_{est}^{(5)}) & -\frac{\partial y^{(5)}}{\partial x^{(1)}} & -\frac{\partial y^{(5)}}{\partial x^{(2)}} & -\frac{\partial y^{(5)}}{\partial x^{(3)}} & -\frac{\partial y^{(5)}}{\partial x^{(4)}} & -\frac{\partial y^{(5)}}{\partial x^{(5)}} & -\frac{\partial y^{(5)}}{\partial x^{(6)}} \\ (y_{ex}^{(6)} - y_{est}^{(6)}) & -\frac{\partial y^{(6)}}{\partial x^{(1)}} & -\frac{\partial y^{(6)}}{\partial x^{(2)}} & -\frac{\partial y^{(6)}}{\partial x^{(3)}} & -\frac{\partial y^{(6)}}{\partial x^{(4)}} & -\frac{\partial y^{(6)}}{\partial x^{(5)}} & -\frac{\partial y^{(6)}}{\partial x^{(6)}} \end{pmatrix}$$

The actual solving of the linear system is carried out efficiently using a triangular factorization of the $\mathbf{A}$ matrix, followed by a simple back-substitution scheme. FORTRAN subroutines LUDCMP and LUBKSB used for these two steps are taken from reference [4]. These same subroutines are also used for inverting the **B** matrix in the tangent evaluation step — equation 4.5.

The final issue related to the corrector sub-algorithm that remains to be discussed, is the convergence criterion. Smith [1] actually employs two criteria and defines the scheme to have converged if either is met. The first is a criterion on the size of the correction vector $\delta\mathbf{u}$. That is, the scheme has converged if

$$|\delta \mathbf{u}| < 1.0\ e - 13$$

Under certain circumstances it is possible to have a tiny correction vector without actually being very close to the solution. For a simple 2-D example, see figure 4-1. Thus, this criterion is not so good and has been left out of the present implementation.

The other criterion that Smith uses, which is retained in this thesis, is a criterion on the residue of the system of equations. It is written explicitly in the following equation :

$$max \left| \frac{y_{est}^{(n)} + \lambda_{pred}(y_{ex}^{(n)} - y_{est}^{(n)}) - y_{pred}^{(n)}}{max(|y_{est}^{(n)}|, |y_{ex}^{(n)}|, |y_{pred}^{(n)}|)} \right| \leq 1.0\ e - 11 \qquad n = 1, \ldots, 6$$

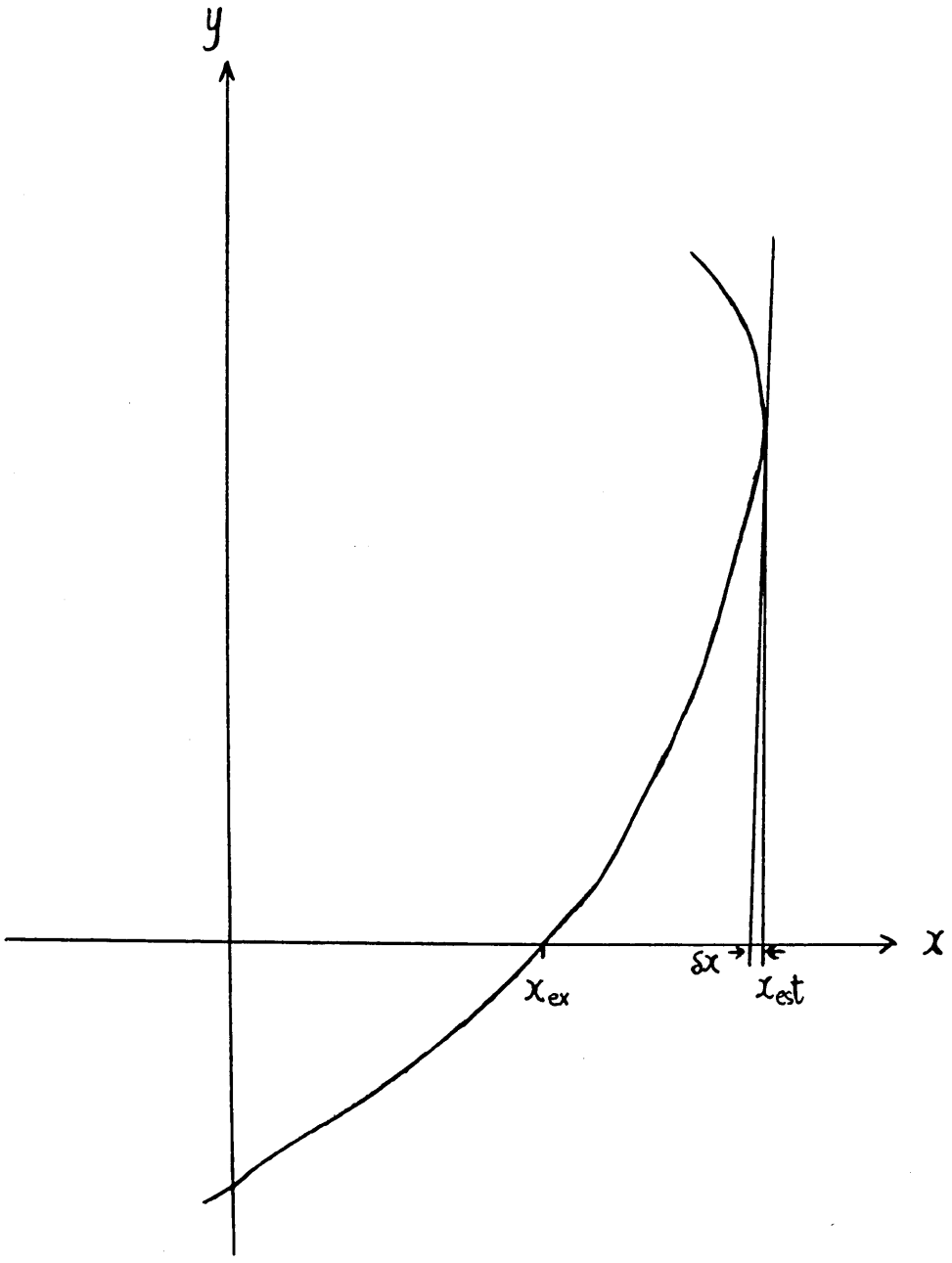

Figure 4-1: Simple 2-D Example Showing Why a Convergence Criterion on the Size of the Correction Vector can be Poor

where the maximum is over n.

This criterion can only be satisfied if the predicted point is virtually a point on the solution curve.

Chapter 5

Problems With the Algorithm

This chapter is concerned with the various problems associated with the use of the curve following algorithm. The problems are sub-divided into two categories :

- Problems which result in algorithm failure.
- Problems which cause inaccurate solutions to be produced.

The expression *algorithm failure* needs to be defined here.

Algorithm failure will be defined as any case in which the algorithm terminates before completing the solution loop. This definition allows for the algorithm to fail but yet produce the desired solution state. If by the time the algorithm terminates prematurely, a $\lambda = 1$ state has been recorded, this may already be the desired solution.

There is another problem that can occur which has been mentioned before, but which shall not be discussed in this chapter. That is the problem of completing the solution loop without getting the desired solution. The solution locus is dependent upon the a priori estimate and the distribution of observation times. If either of these is poorly chosen, the solution loop on which your a priori estimate lies may have no $\lambda = 1$ hyperplane crossing points, or it may have some $\lambda = 1$ hyperplane crossing points none of which are the desired solution. An example is discussed as case #3 in chapter 7. This means that the desired solution lies on a different solution loop to the one on which your a priori estimate lies. The way to overcome this problem is to use the extended method of Smith [1].

5.1 Algorithm Failure

Algorithm failure is very rare when two-body mechanics is the chosen propagation method. In fact, in none of the many real cases that was run with the two-body mechanics propagator did algorithm failure occur. However, there are possible situations which can lead to algorithm failure even with two-body mechanics propagation formulation that was used. One such situation is the existence of epoch states corresponding to hyperbolic orbits on the solution curve. GTDS subroutine BROLYD was used for both twobody and Brouwer-Lyddane propagation in this thesis. This subroutine is unable to handle parabolic or hyperbolic orbits. If the algorithm reaches a point on the solution curve corresponding to a hyperbolic orbit, the corrector step will return to the main algorithm with a non-convergence condition. This in turn will cause the step size to be halved, a new prediction to be made, and the corrector to be tried again. The corrector will repeatedly return a non-convergence condition because the propagator cannot deal with eccentricity > 1. Eventually, the step size will become smaller than the minimum allowed value. At that point, the user will be offered the options of either terminating, restarting at the last corrected point, or restarting at the initial start point in the opposite direction to the one originally chosen. As explained in the section on optional termination in chapter 3, choosing to restart at the last corrected point is not worthwhile under these conditions. There is a formulation of the propagation equations which allows smooth transition between elliptic, parabolic and hyperbolic orbits. This will be discussed in the last chapter.

Fortunately, the existence of hyperbolic solution points on the solution curve is unlikely to occur if the algorithm is being used to determine very low eccentricity orbits and if the chosen a priori estimate also has a low eccentricity. It was noted in the introduction that the Landsat orbits have very low eccentricity, of the order .1.0 e-3.

Another possible situation that could cause algorithm failure even with two-body mechanics propagation is the situation where there is a bifurcation point on the solution curve. That is, the solution curve is precisely at the transition point of

splitting into two loops. This situation is discussed in the first section of chapter 2, schematically illustrated by figure 2-2, and is very unlikely in practice. The shape of the solution curve is dependent, amongst other things, on the a priori estimate chosen (as equation 2.4 indicates). To demonstrate how unusual it would be to have this happen in a real case, I took some real data and varied the epoch estimate very slowly through a transition point ; I was never able to get termination due to existence of a bifurcation point. The plots in figure 5-1 of semi-major axis vs λ show how the solution curve shape changed as the epoch estimate was varied through a transition point. The first plot was produced using the following epoch estimate :

$$a = 7077.8 \text{ km}$$
$$e = 0.001$$
$$i = 100^o$$
$$w = 20^o$$
$$W = 80^o$$
$$m = 130^o$$

The epoch estimates used to produce the second, third, fourth and fifth plots were identical except that the semi-major axes were respectively 7076 km, 7074.238 km, 7074.237 km and 7070 km. These plots indicate that for an epoch estimate with some semi-major axis between 7074.237 km and 7074.238 km, the solution curve may have a bifurcation point.

If the user is unlucky enough to choose an epoch estimate which produces a bifurcation point on the solution curve, the algorithm would reduce the step size near the bifurcation point until it is smaller than the allowed minimum. At that point, the three options mentioned above will be offered.

A good strategy for deciding which option to take when these three are offered is outlined as follows :

1. Check the solution points that have been stored up to the point when the options are offered. If an acceptable solution has already been stored then the obvious choice would be to terminate. Otherwise, continue down this list.

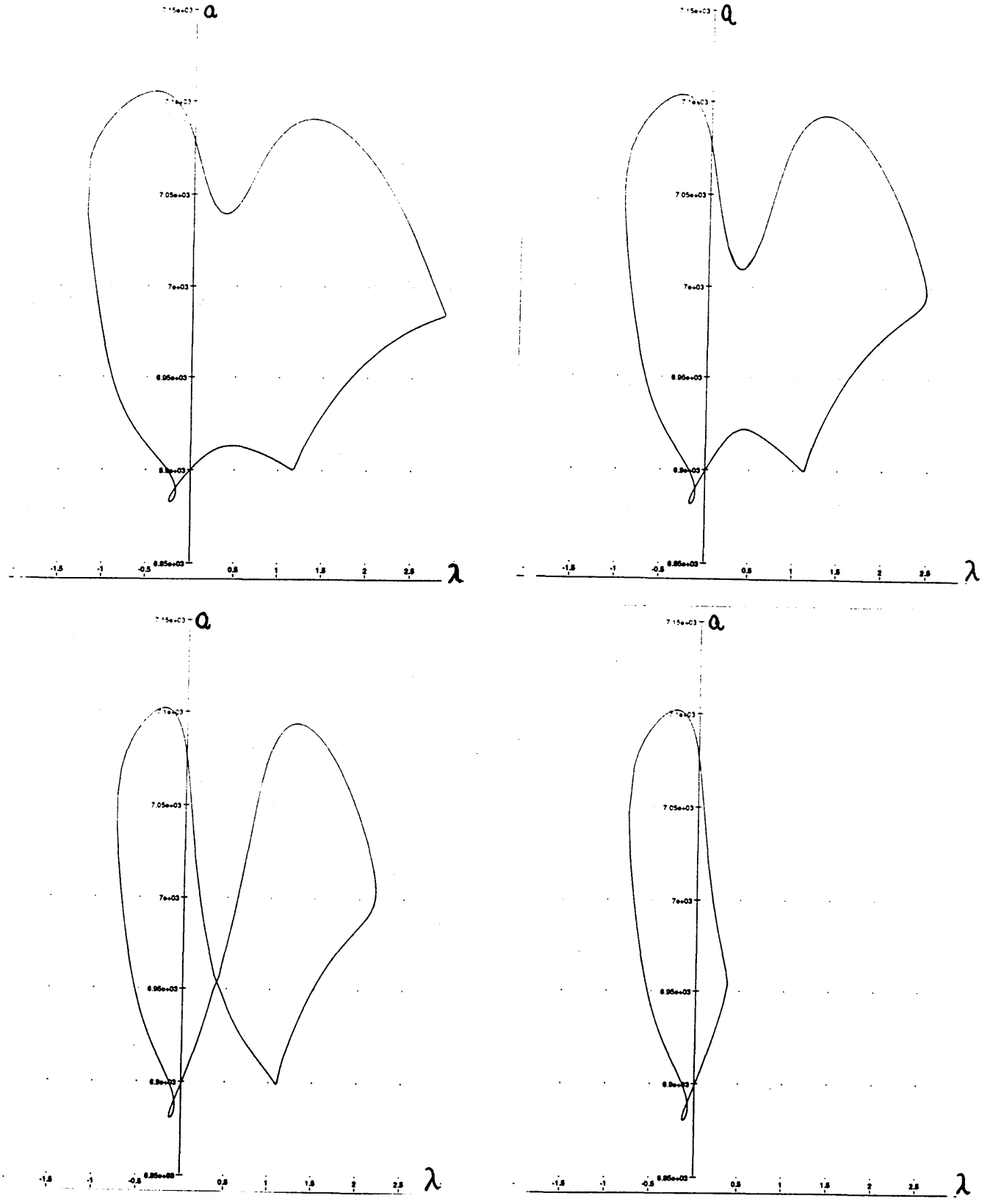

Figure 5-1: Plots Showing the Changing Shape of a Solution Curve as the A Priori Epoch Estimate is Varied in a Real Case

2. Check the number of critical points already stored. If there are ten critical points already stored, then you should try restarting at the last corrected point. If not, continue.

3. Check the eccentricity of the last corrected point. If it is close to 1, then you should try restarting at the initial start point but in the opposite direction to the one originally chosen.

4. If an acceptable solution has not yet been stored, and if the number of critical points stored so far is < 10 and the eccentricity of the last corrected point is not close to 1, then terminate the algorithm and try again with a slightly different a priori estimate.

The situations mentioned above can lead to algorithm failure with either two body mechanics or Brouwer-Lyddane propagation. However algorithm failure is much more common when Brouwer-Lyddane propagation is used. This extra problem with the Brouwer-Lyddane propagation is the fact that there is a singularity at critical inclination $\approx 63.5^o$. Actually, the singularity exists where cosine of the inclination is equal to $\pm 1/\sqrt{5}$. Thus, 116.5^o is also a critical inclination. Note that the singularity at 63.5^o will often lead to algorithm failure because it is typical for the solution curve to cross the inclination $= 63.5^o$ hyperplane at some point, even when the inclination of the desired orbit is quite different. When the algorithm reaches such a crossing point it reacts similarly to the way it reacts to reaching hyperbolic states on the solution curve. That is, the corrector subalgorithm returns a non-convergence condition which causes the step size to be halved until it is smaller than the minimum allowed value. At that point the optional termination is offered along with the restart options.

NOTE A neat technique for avoiding this singularity has been implemented late in the proceedings of this thesis. That technique is discussed in the next chapter.

5.2 Inaccurate Results

There are three kinds of error that can lead to inaccuracy in the solutions obtained : measurement errors, round-off errors, and modeling errors. Round-off error is minimized in this algorithm by using high precision storage of variables. It is safe to say that inaccuracy in the solution caused by round-off errors is negligible compared with that caused by other sources. This is confirmed by runs in which range data is simulated so that modeling and measurement errors are not present.

The amount of solution inaccuracy arising out of measurement error depends on the quality of the instrumentation, the location of the tracking station and on the distribution of measurement times. For example, taking six measurements over a very short time span will cause small errors in measurement to be severely amplified in the solution. This manifests itself as ill-conditioning of the Jacobian matrix $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$. One potentially important area of future research would be the investigation of how the Jacobian matrix depends on measurement time distribution, tracking station location, and satellite orbit. Inaccuracies in the measurement times (tracking station clock errors) are also potentially important, particularly for short arc fits (closely spaced measurements).

The source of inaccuracy that would be easiest to improve is the modeling of the dynamics of the problem. The solution $\mathbf{x}_{ex}$ that is produced by the algorithm will be such that :

$$\mathbf{f}(\mathbf{x}_{ex}) = \mathbf{y}_{ex}$$

Clearly the greater the dynamics modeling inaccuracy contained in $\mathbf{f}$, the greater will be the inaccuracy in the solution produced.

In order to keep the problem simple enough to be run on a VAXstation 3100, simple propagation models such as two-body mechanics and Brouwer-Lyddane were used. The two-body model ignores the gravitational perturbations caused by the sun and the moon, the solar radiation pressure, the effects of atmospheric drag and the zonal and tesseral harmonics of the Earth's gravitational field. The Brouwer-Lyddane propagation model is an improvement because it accounts for the effects of the first

four zonal harmonics. The solution improvement resulting from this better modeling in demonstrated in chapter 7.

The error resulting from a lack of atmospheric drag modeling would be greater for low altitude satellites, particularly in periods of high solar activity when the atmosphere becomes more dense. The error resulting from not including the solar and lunar gravitational perturbations would be greater for high altitude satellites such as geostationary satellites.

Another potential source of solution error is inaccuracy in the following data :

- the tracking station Earth-fixed coordinates
- the right ascension of Greenwich at epoch
- the values of certain constants such as μ, the gravitational constant of the Earth, and ω, the rotational rate of the Earth

These quantities are generally known to high precision and so should not be a significant problem.

The longer the elapsed time between epoch and a measurement, the greater will be the solution error caused by all of these modeling errors. Therefore it is better to make measurements relatively soon after the desired epoch time. However we cannot make all of the measurements too close together or we run into the problem of ill-conditioning mentioned above.

Clearly, the distribution of times at which measurements are taken is something to which some attention should be paid. The measurement times should be sufficiently far apart to prevent significant inaccuracy due to measurement error, yet they should be close enough to the epoch time, to prevent significant errors due to inaccuracy in the gravitational field and atmospheric drag modeling. We are further restricted by the fact that the satellite will only be in view of our tracking station a small percentage of the time. A measurement strategy that has produced fairly good results with this algorithm for the Landsat orbit using a northern tracking station, is to use three times spaced one minute apart in one pass of the satellite, followed by three more times spaced one minute apart in the next pass of the satellite. A better distribution might

be to take the six measurements at the beginning, middle and end of consecutive passes. The advantage of a high latitude station for Landsat tracking, would be that the Landsat orbit is almost polar and so would be visible for some part of most, if not all, of its orbits.

The effect of having measurement times that are too close together is demonstrated by case #1 in chapter 7. The effect of having measurement times that are too far away from the epoch time is demonstrated by case #3 in chapter 7.

Chapter 6

Removing The Brouwer-Lyddane Singularity at Critical Inclination

As explained in chapter 5, the Brouwer-Lyddane propagation model has a singularity at the critical inclination of approximately 63.5°. If we would like to be able to use the algorithm for general preliminary orbit determination, we need to be sure that this singularity is removed.

A clever technique for avoiding the singularity in inclination at 63.5° was devised by the former Technical Director of the Naval Space Surveillance System (NAVSPASUR), Richard H. Smith [5] and is implemented in the PPT General Perturbation Satellite Theory. The singularity arises because the calculation of the Brouwer-Lyddane osculating elements involves terms including the factor :

$$\frac{1}{1 - 5\cos^2 i} \tag{6.1}$$

The trick is to arbitrarily replace this factor with the following :

$$\frac{1 - e^{-100x^2}}{x} \tag{6.2}$$

where x is given by :

$$x = 1 - 5\cos^2 i$$

The plot of the replacement factor given by equation 6.2 vs inclination is compared with the plot of the exact factor given by equation 6.1 in figure 6-1. The replacement factor is virtually equal to the exact factor everywhere except very close to the critical inclination, where the replacement factor remains bounded instead of having the singularity of the exact factor. A plot of the replacement factor divided by the real factor vs inclination is shown by figure 6-2.

The effectiveness of using the replacement factor is illustrated by the plots in figures 6-3 and 6-4. Figure 6-3 shows a plot of inclination vs λ for a real case using the exact factor. Figure 6-4 shows the corresponding plot when the replacement factor is used. Using the exact factor forced the algorithm to terminate without completing the loop because of the singularity at 63.5^o, whereas using the replacement factor got rid of the singularity and allowed the loop to be completed.

This trick for getting rid of the singularity in inclination was brought to my attention by Paul Cefola towards the end of the time allowed for this thesis and so was only included at the end. It represents a significant improvement to the algorithm. Smith [1] does not even mention the problem.

Note that this trick will also work for the singularity at inclination $\approx 116.5°$.

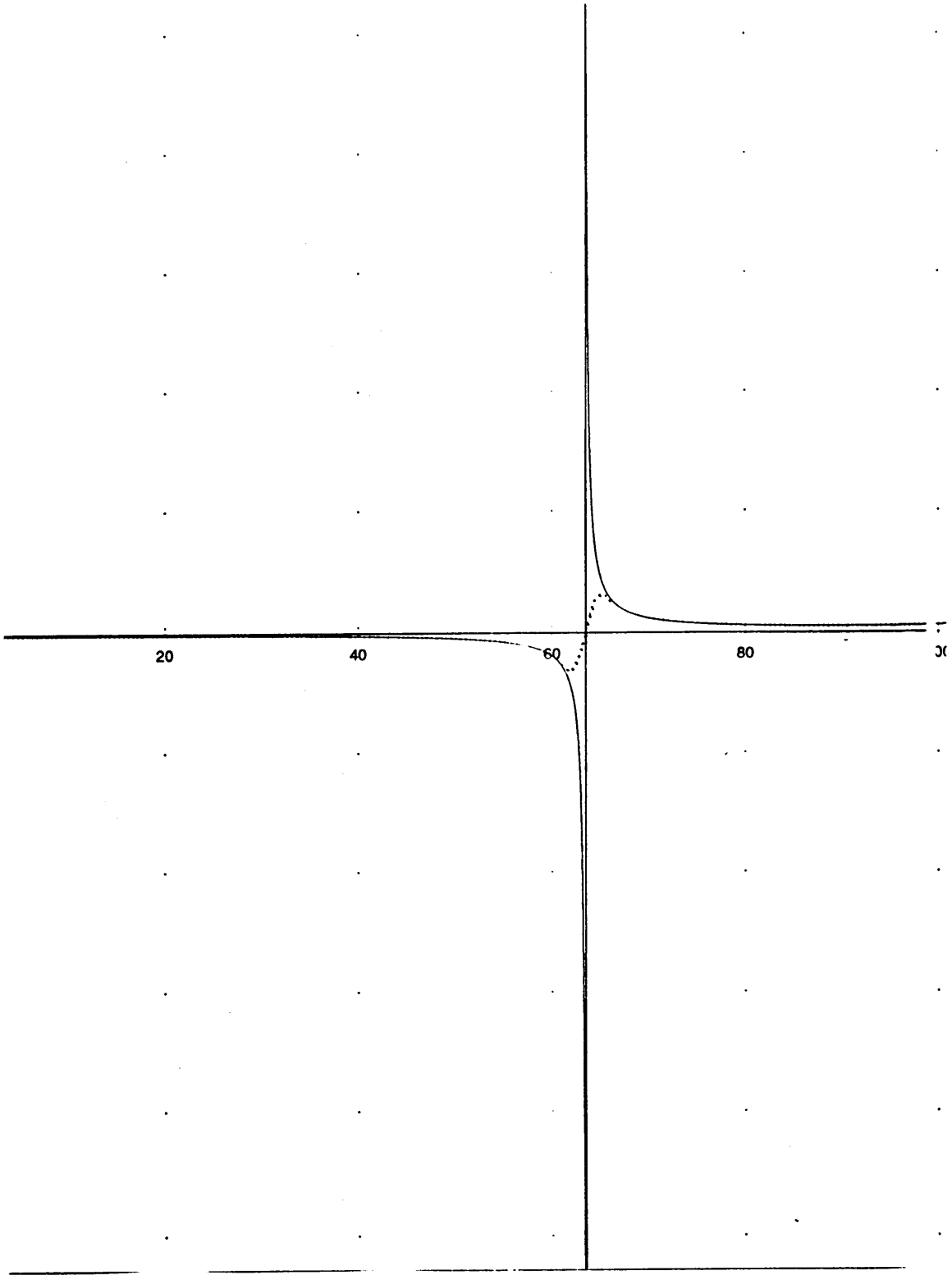

Figure 6-1: Plots of Exact and Replacement Factor vs Inclination

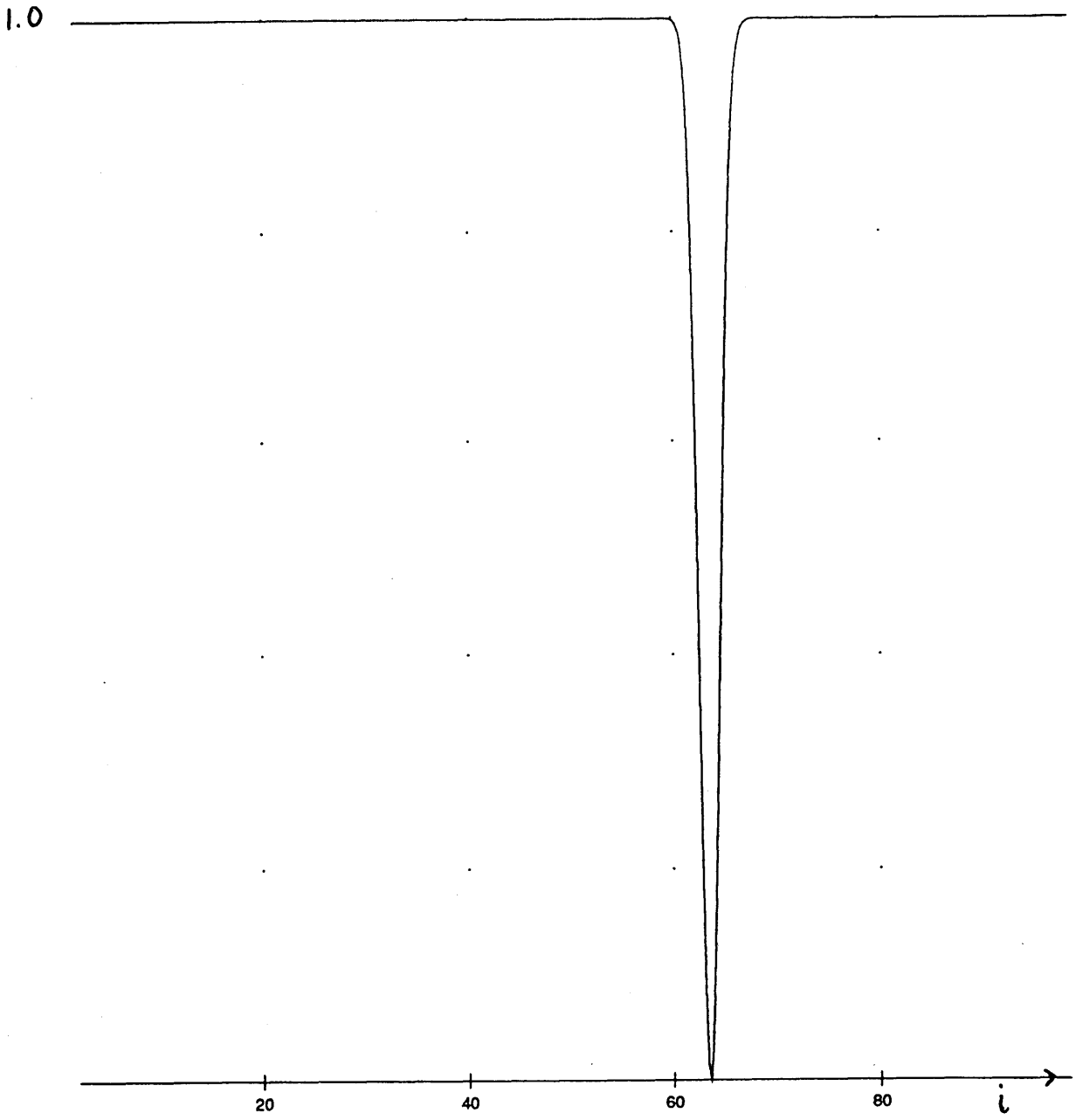

Figure 6-2: Plot of Replacement Function Divided by Exact Function vs Inclination

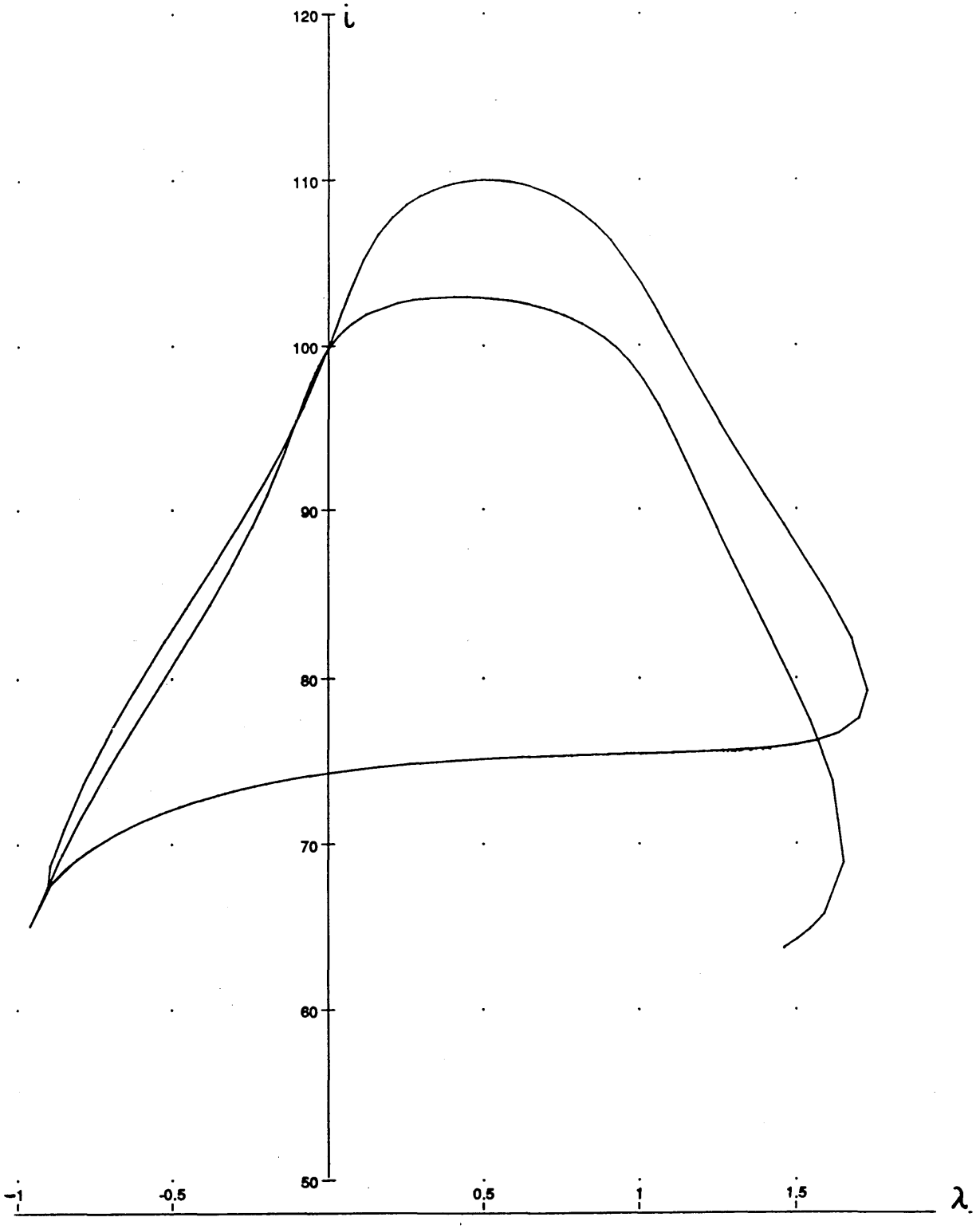

Figure 6-3: Plot of Inclination vs λ Using the Exact Factor in a Real Case

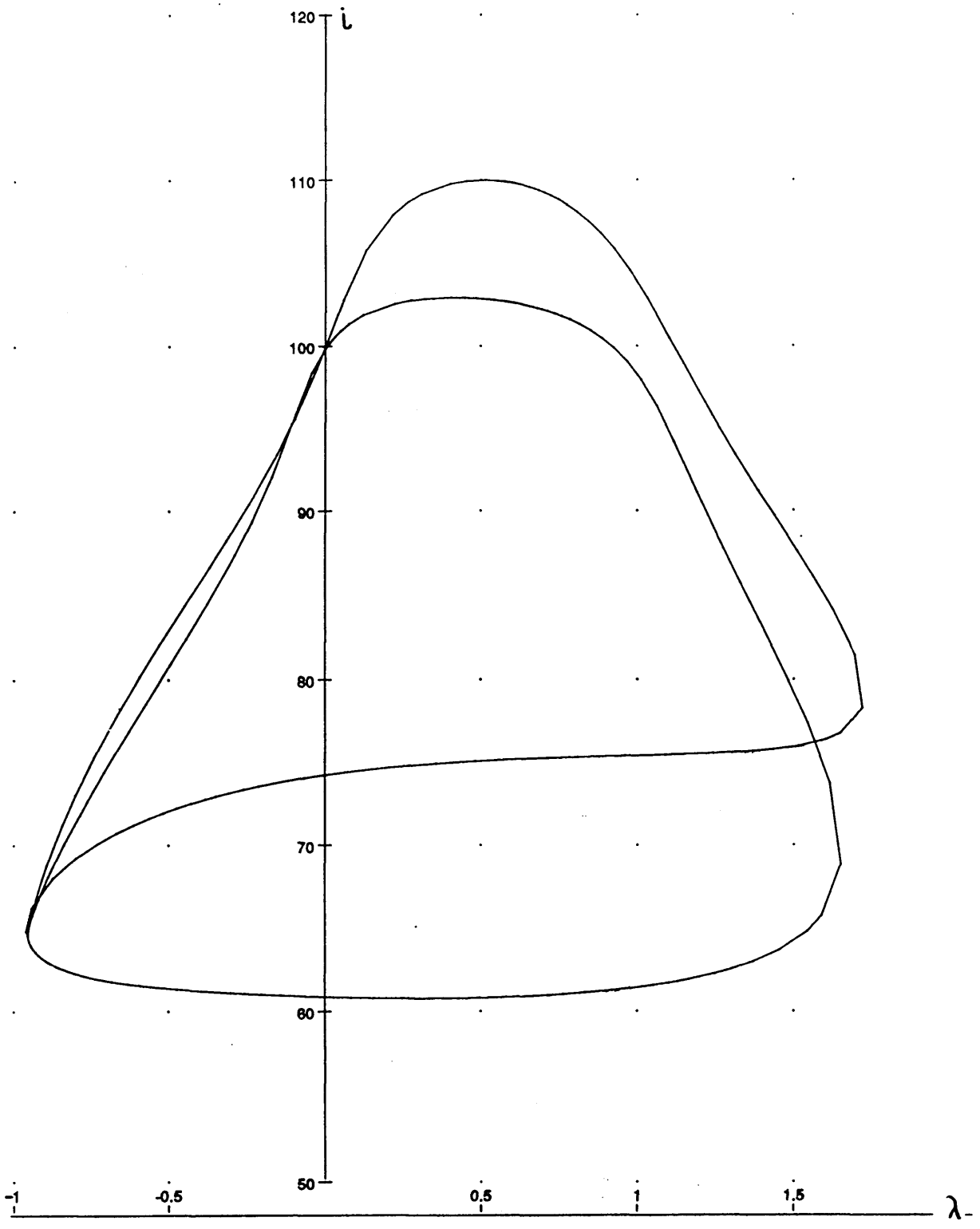

Figure 6-4: Plot of Inclination vs λ Using the Replacement Factor in a Real Case

Chapter 7

Application to the Landsat Satellite

The effectiveness of the algorithm in determining the epoch state of a Landsat satellite was evaluated by running the program using real Landsat 4 tracking data. The solutions produced using both two-body mechanics and Brouwer-Lyddane orbit propagation models are compared with very accurate solutions produced at The Charles Stark Draper Laboratory in Cambridge, Massachusetts. The tracking data used was provided by the NASA Goddard Space Flight Center (GSFC).

The accurate solutions from Draper Laboratory were produced using a batch least squares fit to the tracking data. In general, the tracking data used consisted of a combination of two to three hundred range, range-rate and angular measurements made from the following tracking stations : Greenbelt, MD ; Goldstone, CA ; Merritt Island, FL ; Guam ; Bermuda ; Madrid, Spain ; Fairbanks, AK. Generally, observations were made from about six passes of the satellite spanning about two days.

The standard deviations for position coordinates in the solutions produced by Draper were typically about 50 meters and the standard deviations for the velocities were typically about 50 millimeters per second.

The orbit propagation method used at Draper was a Cowell numerical integration technique with a 60 second step size. The dynamic modeling included the following :

1. The gravity model used was GEM 9, a 21-by-21 potential field model.

2. Atmospheric drag was included using the Harris-Priester ($F_{10.7} = 175$) atmospheric density model.

3. GTDS Solar/Lunar/Planetary (SLP) files were used to compute the gravitational perturbations caused by the Sun and the Moon.

4. The effects of solar radiation pressure were modeled.

Some notation that will be used in this chapter is necessary here. Solution states will be given in true of reference Cartesian coordinates and corresponding mean Kepler elements. The position coordinates will be denoted by x, y and z, and the corresponding velocity coordinates will be denoted by xv, yv and zv. The a priori estimates will be given in mean Kepler elements. As before, the semi-major axis will be denoted by a, the eccentricity by e, the inclination by i, the longitude of the ascending node by w, the argument of perigee by W, and the mean anomaly by m. The observation times are given in seconds after the epoch time.

Five real test cases will be presented in this chapter. In each case the tracking data used came from the Fairbanks station. That station has the following geodetic parameters :

$$\text{latitude} \approx 64.972^{o}$$
$$\text{longitude} \approx 212.487^{o}$$
$$\text{height} \approx 334.39 \text{ meters}$$

NOTE: EOSAT planned to track Landsat 6 from Fairbanks, Alaska at the time work was begun on this thesis. Recent plans to move the tracker to Norman, Oklahoma are not addressed in this work.

7.1 Landsat 4 Test Cases

TEST CASE #1

In this case the epoch time was 00:00 hours on the 10^{th} of November 1982. The right ascension of Greenwich was $\approx 48.8262^{\circ}$ at that time. The exact epoch state as calculated at Draper was :

x = -4411.7615 km	a = 7077.8449 km
y = -265.5134 km	e = 1.8117 e-4
z = -5542.8760 km	i = 98.2430 degrees
xv = 5.5846 km/s	w = 13.9188 degrees
yv = 2.0622 km/s	W = 78.0268 degrees
zv = -4.5471 km/s	m = 154.0688 degrees

The epoch estimate used was :

a = 7050 km
e = 0.001
i = 100 degrees
w = 10 degrees
W = 100 degrees
m = 130 degrees

Using the following observations,

times	ranges
21060	835.7509 km
21120	1067.7043 km
21180	1397.5961 km
27120	1892.3992 km
27180	2248.5968 km
27230	2559.4028 km

the solutions produced by the algorithm using each of the propagation models were as follows :

two-body	Brouwer-Lyddane
-4693.6247 km	-4415.7695 km
-226.6133 km	-267.1547 km
-5291.1003 km	-5540.0795 km
5.4572 km/s	5.5819 km/s
1.4940 km/s	2.0612 km/s
-4.9277 km/s	-4.5504 km/s

The amount by which these solutions differ from the accurate Draper solution is represented by the following :

two-body error	Brouwer-Lyddane error
281.9 km	4.0 km
35.9 km	1.6 km
251.8 km	2.8 km
127.4 m/s	2.7 m/s
568.2 m/s	1.0 m/s
380.6 m/s	3.3 m/s

The effect of having all six measurements too close together was demonstrated by running this same case but with the following observations :

times	ranges
21060	835.7509 km
21120	1067.7043 km
21180	1397.5961 km
21240	1770.5617 km
21300	2163.2992 km
21360	2565.4879 km

Using these measurements, the solution curve did not cross the $\lambda = 1$ with either two-body or Brouwer-Lyddane propagation. The change in performance of the algorithm

caused by taking all six observations from a single pass of the satellite separated by one minute each, instead of taking three observations separated by one minute each from one pass, and then three more separated by a minute each from the next pass, is clearly very substantial. This confirms the need to pay attention to the distribution of observation times.

TEST CASE #2

In this case the epoch time was 00:00 hours on the 13^{th} of November 1982. The right ascension of Greenwich was $\approx 51.7828^{o}$ at that time. The exact epoch state as calculated at Draper was :

x = -3035.5649 km	a = 7077.9005 km
y = -1848.0000 km	e = 1.8313 e-4
z = 6115.9519 km	i = 98.2425 degrees
xv = -6.4306 km/s	w = 16.8868 degrees
yv = -1.4063 km/s	W = 81.5833 degrees
zv = -3.6078 km/s	m = 37.4597 degrees

The epoch estimate used was :

a = 7070 km
e = 0.001
i = 100 degrees
w = 20 degrees
W = 80 degrees
m = 40 degrees

Using the following observations,

times	ranges
17100	1377.1289 km
17160	1614.6082 km
17220	1920.4452 km

22680	1478.2112 km
22740	1120.6229 km
22800	834.1570 km

the solutions produced by the algorithm using each of the propagation models were as follows :

two-body	Brouwer-Lyddane
-2967.7452 km	-3036.1244 km
-1693.2593 km	-1848.1642 km
6197.8204 km	6116.2554 km
-6.5219 km/s	-6.4300 km/s
-1.3215 km/s	-1.4069 km/s
-3.4735 km/s	-3.6075 km/s

The amount by which these solutions differ from the accurate Draper solution is represented by the following :

two-body error	Brouwer-Lyddane error
67.8 km	0.5 km
154.7 km	0.2 km
81.9 km	0.3 km
91.3 m/s	0.6 m/s
84.8 m/s	0.6 m/s
134.3 m/s	0.3 m/s

TEST CASE #3

In this case the epoch time was 00:00 hours on the 16^{th} of November 1982. The right ascension of Greenwich was $\approx 54.7398^{\circ}$ at that time. The exact epoch state as calculated at Draper was :

x = 6661.3348 km	a = 7077.8502 km
y = 2294.3875 km	e = 1.6676 e-4

z = 721.2994 km	i = 98.2425 degrees
xv = -0.3726 km/s	w = 19.8558 degrees
yv = -1.2715 km/s	W = 61.5230 degrees
zv = 7.3866 km/s	m = 304.5261 degrees

The epoch estimate used was :

a = 7080 km
e = 0.001
i = 100 degrees
w = 20 degrees
W = 60 degrees
m = 300 degrees

Using the following observations,

times	ranges
18720	1350.1219 km
18780	1110.4840 km
18840	1008.8227 km
-64140	1168.6036 km
-64200	870.7931 km
-64260	723.0790 km

the algorithm failed to produce the desired solution with either of the propagation models.

This case is an example of having the measurement times too far from the required epoch. Running this case with two-body mechanics propagation produced a curve which never crossed the $\lambda = 1$ hyperplane. Running the case with Brouwer-Lyddane gave a curve which did cross the $\lambda = 1$ hyperplane twice but neither of those solutions was the desired one.

TEST CASE #4

In this case the epoch time was 00:00 hours on the 8^{th} of November 1982. The right ascension of Greenwich was $\approx 46.8549°$ at that time. The exact epoch state as calculated at Draper was :

x = -6898.0071 km	a = 7078.0078 km
y = -1320.3564 km	e = 1.7567 e-4
z = -933.9781 km	i = 98.2427 degrees
xv = 0.7486 km/s	w = 11.9405 degrees
yv = 1.2473 km/s	W = 88.7248 degrees
zv = -7.3592 km/s	m = 98.7843 degrees

The epoch estimate used was :

a = 7050 km
e = 0.001
i = 100 degrees
w = 10 degrees
W = 100 degrees
m = 80 degrees

Using the following observations,

times	ranges
15960	1419.7035 km
16020	1584.5052 km
16080	1837.9442 km
21600	1275.6315 km
21660	956.1996 km
21720	757.2672 km

the solutions produced by the algorithm using each of the propagation models were as follows :

two-body	Brouwer-Lyddane
-6911.4285 km	-6899.2410 km
-1303.0816 km	-1321.3327 km
-885.1156 km	-931.7927 km
0.7064 km/s	0.7465 km/s
1.2212 km/s	1.2463 km/s
-7.3617 km/s	-7.3583 km/s

The amount by which these solutions differ from the accurate Draper solution is represented by the following :

two-body error	Brouwer-Lyddane error
13.4 km	1.2 km
17.3 km	1.0 km
48.9 km	2.2 km
42.2 m/s	2.1 m/s
26.1 m/s	1.0 m/s
2.5 m/s	0.9 m/s

TEST CASE #5

In this case the epoch time was 00:00 hours on the 17^{th} of November 1982. The right ascension of Greenwich was $\approx 55.7253°$ at that time. The exact epoch state as calculated at Draper was :

x = -5992.2672 km	a = 7077.8455 km
y = -1761.7529 km	e = 1.5872 e-4
z = -3350.3145 km	i = 98.2409 degrees
xv = 3.0061 km/s	w = 20.8421 degrees
yv = 2.1543 km/s	W = 67.4594 degrees
zv = -6.5209 km/s	m = 140.9419 degrees

The epoch estimate used was :

a = 7070 km

e = 0.001

i = 100 degrees

w = 20 degrees

W = 80 degrees

m = 130 degrees

Using the following observations,

times	ranges
15600	1429.4088 km
15660	1547.8971 km
15720	1766.8760 km
21300	1065.9566 km
21360	820.4502 km
21420	763.8133 km

the solutions produced by the algorithm using each of the propagation models were as follows :

two-body	Brouwer-Lyddane
N	-5994.1642 km
O	-1761.9712 km
N	-3347.8987 km
E	3.0042 km/s
	2.1522 km/s
	-6.5218 km/s

The amount by which this solution differ from the accurate Draper solution is represented by the following :

Brouwer-Lyddane error
1.9 km
0.2 km

2.4 km

1.9 m/s

2.1 m/s

0.9 m/s

Running this case with two-body mechanics propagation produced a solution curve which never intercepted the $\lambda = 1$ hyperplane. Running it with Brouwer-Lyddane propagation produced a curve which not only crossed the $\lambda = 1$ hyperplane more than once, but recovered the desired solution at one of those crossings. This case is an illustration of the greater accuracy of the Brouwer-Lyddane propagation model.

7.2 General Comments

The above results show that the algorithm is effective at determining the epoch state of a Landsat satellite using only six range measurements from a single Earth-based tracker.

The accuracy of that epoch state is very dependent on the accuracy of the propagation model used. Brouwer-Lyddane propagation is much more accurate than two-body mechanics.

The strategy of taking three measurements separated by one minute during one pass of the satellite, then taking three more measurements separated by one minute during the next pass yields good performance.

Measurements were taken during the fourth and fifth passes after epoch in case #1, and during the third and fourth passes after epoch in cases #2, #4 and #5. The accuracies in position coordinates for cases #2, #4 and #5 were about 1 km with Brouwer-Lyddane and about 30 km with two-body mechanics. The accuracies in velocity coordinates for these cases were about 1 m/s with Brouwer-Lyddane and about 30 m/s with two-body mechanics. The accuracies for case #1 were significantly worse. Position coordinates were accurate to about 3 km with Brouwer-Lyddane and about 200 km with two-body mechanics, while velocity coordinates were accurate to about 3 m/s with Brouwer-Lyddane and about 350 m/s with two-body mechanics. This

evidence may be explained by the fact that in case #1 longer times were allowed for the errors in dynamic modeling in the propagators to manifest themselves as solution inaccuracy. We would therefore expect significant improvement in our solutions if epoch was chosen so that the measurements were taken during the first and second satellite passes after epoch.

Chapter 8

Use of the Software

This chapter addresses the use of the algorithm developed for this thesis and suggests a method for getting maximum efficiency from it.

When processing real data we must choose whether to select two-body mechanics or Brouwer-Lyddane as the propagation dynamics model. The model chosen should be the result of trading off the greater accuracy of the Brouwer-Lyddane method against the greater speed of the two-body mechanics method. If a relatively inaccurate solution is acceptable to the user, then it is better to use the two-body mechanics propagation method and then screen out the desired solution from the set of solutions obtained. If a relatively accurate solution is required, then Brouwer-Lyddane will have to be used. Note that although the only two propagation models included in this algorithm are two body mechanics and Brouwer-Lyddane, other, more accurate models could be added if necessary.

There is a strategy that can be used for obtaining the accuracy of the Brouwer-Lyddane propagation, without sacrificing too much computational speed. That strategy is outlined as follows :

1. Run the data using two-body propagation.

2. Then screen out the desired but inaccurate solution assuming that the desired solution lies on the same loop as the a priori estimate chosen.

3. Then run the data using the Brouwer-Lyddane propagator with the inaccurate

two-body solution as the a priori estimate.

4. Stop the program immediately after the $\lambda = 1$ is crossed for the first time. The user will know when $\lambda = 1$ because after each corrected point is computed, its λ value is written to the screen. The single solution stored, will be the relatively accurate Brouwer-Lyddane solution, corresponding to the relatively inaccurate two body mechanics solution, obtained from the two-body part of the operation, and used as the a priori estimate for the Brouwer-Lyddane part.

If another, more accurate and more time consuming propagation model was added, the strategy would then be to take the Brouwer-Lyddane solution and use it as the a priori estimate for that new model.

Note that in step 2 of the suggested strategy it is assumed that the solution loop completed by running the two-body propagation model contains the desired solution. If this is not the case, we have to abandon the suggested strategy. However, what we should do at that point is to run the initial a priori estimate with the more accurate Brouwer-Lyddane propagator. It is possible that the desired solution *will* lie on the solution loop produced then. An example of this is discussed in case #5 of the previous chapter.

If we were concerned with maximizing the efficiency of the algorithm (minimizing its run time), we could remove most of the writing to files that is done in the present software implementation. Using the strategy suggested above as opposed to running Brouwer-Lyddane directly on the initial epoch estimate is another time saving approach. The following figures indicate how effective implementing these two time saving techniques can be :

Running case #1 directly with Brouwer-Lyddane propagation took $7\frac{1}{4}$ minutes
Running case #1 using the suggested strategy and retaining the unnecessary writing to files took 4 minutes.
Running case #1 using the suggested strategy and without the unnecessary writing to files took $2\frac{3}{4}$ minutes.

Running case #2 directly with Brouwer-Lyddane propagation took 13 minutes
Running case #2 using the suggested strategy and retaining the unnecessary writing to files took $3\frac{1}{2}$ minutes.
Running case #2 using the suggested strategy and without the unnecessary writing to files took $2\frac{3}{4}$ minutes.

The above times *are not* CPU times, they are run turnaround times in a heavily used time sharing environment. The hardware used was a VAXstation 3100.

Chapter 9

Conclusions

The algorithm in its current form is a useful tool for preliminary orbit determination. An immediate application could be the preliminary orbit determination of the Landsat 6 satellite. The only data it requires consists of :

1. Six range observations from a single Earth-based tracker, and the corresponding six observation times
2. The geodetic latitude, longitude and height of the Earth-based tracker.
3. The right ascension of Greenwich at the desired epoch time.
4. An a priori estimate of the epoch state to be determined.

In order to work satisfactorily a reasonably good a priori estimate, such as would certainly be available in the Landsat 6 case, is required.

The accuracy of the solution produced depends on, amongst other things, the accuracy of the observation data, the tracker geodetic parameters, and the right ascension of Greenwich at epoch.

Solution accuracy also depends on the distribution of observation times. A detailed study of this dependence would be very useful. The real test cases that were run for this research indicate the following :

- Taking all six observations at one minute intervals during a single pass of the satellite is a bad strategy.

- Taking three observations at one minute intervals during one pass of the satellite and then taking the next three observations at one minute intervals during the next pass is a good strategy.

- Using observations from the 3^{rd} and 4^{th} passes produced significantly more accurate solutions than those produced by using observations from the 4^{th} and 5^{th} passes.

- Choosing epoch such that the measurements correspond to the 1^{st} and 2^{nd} passes of the satellite would produce even more accurate solutions.

- The accuracy of the solution produced by this algorithm depends on the propagation model used. Brouwer-Lyddane produced much more accurate solutions than two-body mechanics. The amount of solution improvement resulting from using a better propagator is a function of the distribution of measurement times. That is, the closer to the measurement times you choose epoch to be, the lesser will be the improvement made by using a better propagation model.

- The algorithm produced during this research is a useful tool which has immediate application to the Landsat 6 preliminary orbit determination problem.

Chapter 10

Recommendations for Future Work

The algorithm in its present state is a very useful tool for the preliminary orbit determination of a limited range of orbits, in particular, the Landsat orbits. However, there are a number of improvements that could be implemented that would greatly expand the usefulness of the algorithm by extending the range of orbits it could be used to determine.

10.1 The Extended Six-Level Scheme

One major improvement would be to extend the method to the six-level scheme discussed by Smith [1]. As it stands the algorithm can only complete the solution loop upon which the chosen a priori estimate lies. Hence, only solutions that lie on that loop will be recorded. If the a priori estimate is a relatively good one, then the solution which we seek will lie on the same loop as the a priori estimate. In the case of Landsat 6, if the launch goes according to plan, a relatively good a priori estimate will be available and so the algorithm is good enough as it stands provided that a sensible measurement time distribution is used. This is supported by the results presented in chapter 7. In fact, in most cases where friendly man made satellites are concerned, a good a priori estimate should be available. However, if we consider a situation in

which the satellite launch goes wrong and we have no good a priori estimate, then the algorithm as it stands may be inadequate. With the six-level scheme, the method becomes global, that is, regardless of the a priori estimate used, the algorithm will store all the possible solutions. Therefore, in the case of a misfired launch, the six-level scheme would still record the desired solution provided that tracking data was available. The problem then might be to screen the correct solution from amongst the solutions stored.

Another example of when a good a priori estimate would not be available, is in the case where you were trying to determine the flight path of an enemy ballistic missile. For that situation we would again need the extended six-level scheme. The algorithm could then be further extended to allow propagation of the determined epoch state to some future time so that coordinates for an interception could be calculated. The basic problem we could anticipate for this kind of application would be the fact that the six observation times would have to be close together and it has already been noted that this can lead to solution inaccuracy due to measurement error. However, the ranges would be very short compared to the ranges used in satellite orbit determination and so there should be very little error in the range measurements.

10.2 Allowing for Hyperbolic States

The next important improvement that could be made to this algorithm would be to modify the orbit propagation models so that they can deal with hyperbolic orbit states. The Landsat orbits are very low eccentricity orbits and so if we use a good a priori estimate that also has very low eccentricity, we would not expect to find any hyperbolic states lying on the solution curve, although that situation is not impossible. Figure 10-1 demonstrates this. This figure shows a plot of eccentricity vs lambda for a real Landsat data case. The very accurate approximation to the real solution that was produced by the Cowell propagator used by Draper and the a priori estimate used are shown below.

real solution	a priori estimate
a = 7082.8470 km	a = 7077 km
e = 0.0009986	e = 0.001
i = 98.2380 degrees	i = 100 degrees
w = 20.8376 degrees	w = 20 degrees
W = 75.0422 degrees	W = 80 degrees
m = 133.4049 degrees	m = 130 degrees

The figure shows that, although the real eccentricity is very low, and a relatively good a priori estimate, also with very low eccentricity is used, the solution curve still comes very close to crossing the eccentricity = 1 hyperplane. The maximum value of eccentricity on the plot is ≈ 0.986. Behavior like this is unusual for Landsat data processing. However, if we hope to use this algorithm for more general orbit determination, in which the real solution may have medium or high eccentricity, then it would be necessary to modify the propagation models so that they will allow hyperbolic orbit states on the solution curve. One way to do this would be to use the Universal formulation for two-body mechanics discussed in [9].

10.3 Sensitivity Study

An area for useful future work would be the study of the sensitivity of the algorithm to the distribution of observation times. Of particular interest would be the short arc case. That is, the case in which all measurements are taken over a short period of time, e. g. , from a single pass. Determining the path of a ballistic missile falls into this category. This kind of application would require a high precision propagator.

Sensitivity to tracking station location and to the type of orbit (e. g. , geostationary orbit) are other areas of interest that would represent useful future work.

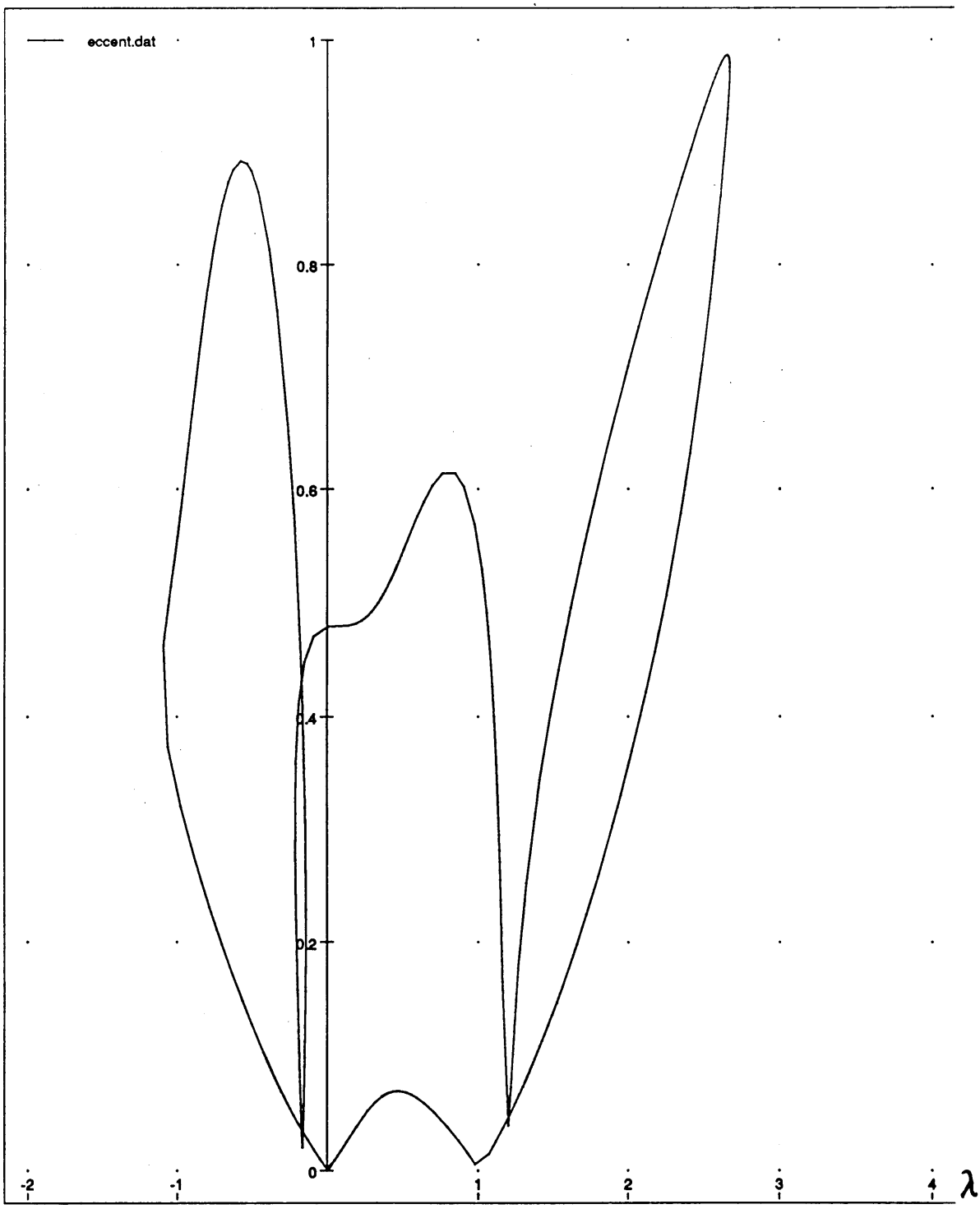

Figure 10-1: A Plot Showing That Eccentricity on the Solution Curve Can be Large Even if the A Priori Estimate and the Real Solution Have Very Low Eccentricities

10.4 Atmospheric Drag Modeling

Another improvement that could be made to this algorithm would be to include some atmospheric drag modeling in the dynamics of the problem. This would serve to improve the accuracy of the solutions obtained from the algorithm in general. For high altitude orbits, such as geostationary orbit, or even for medium altitude orbits, such as the Landsat orbits, the improvement would not be great, since over the period of time spanned by the observations, atmospheric drag does not have a marked effect. However, for a lower altitude orbit, the improvement made by including a drag model may be worthwhile. A major problem with drag modeling is that density profile of the atmosphere is difficult to predict. It depends strongly on solar activity.

10.5 Improved Gravitational Field Modeling

Clearly the algorithm could be improved, in terms of producing more accurate solutions, by implementing a better model of the gravitational forces on the satellite. That is, we could use a propagation model that takes account of more of the zonal and tesseral harmonics of the Earth's gravitational field. Also we could include the effect of the gravitational attraction of the Sun and the Moon. Of course, there is a trade-off between the greater accuracy and the increased run time that would result from an improved gravity model. Chapter 8 provides a strategy on how best to optimize the trade-off between run time and solution accuracy. That strategy is basically to use the solution obtained from running the data with the less accurate propagation model, as the a priori estimate for data processing with the more accurate propagation model.

10.6 Application to Non-Terrestrial Satellites

In this final section, the possibility of using this preliminary orbit determination method for determining the orbits of satellites about other planets such as Mars is considered. In order to use this algorithm, certain modifications would be necessary.

The basic method could remain unchanged, but any parameters that are local to

the planet Earth would have to be replaced by the corresponding parameters of Mars. The two body mechanics propagator would need to replace the values of the Earth's angular rate, its gravitational parameter, μ, and its size and shape parameters with the corresponding Mars parameters.

Atmospheric modeling, solar radiation pressure, gravitational perturbations due to external bodies and zonal and tesseral harmonics would have to be altered to model the conditions on Mars.

The extension to the six-level method, and the modification to allow hyperbolic orbit states on the solution curve, would be unaffected by the change of planet.

Bibliography

[1] Smith, R. L., and Huang, C. Y., 'Study of a Homotopy Continuation Method for Early Orbit Determination With the Tracking and Data Relay Satellite System (TDRSS)', NASA Technical Memorandum 86230.

[2] Smith, R. L., and Huang, C. Y., 'A Homotopy Continuation Method for General Preliminary Orbit Determination and Special Application to the Tracking and Data Relay Satellite System', Paper No. 85-0215 , Presented at the American Institute of Aeronautics and Astronautics 23^{rd} Aerospace Sciences Meeting, Reno, Nevada, January 14-17, 1985.

[3] Escobal, P. R., 'Methods of Orbit Determination', New York, John Wiley and Sons Incorporated, 1965.

[4] Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T., 'Numerical Recipes', Cambridge University Press, 1986.

[5] Richard H. Smith, 1983, private communication with Felix R. Hoots, February, 1983.

[6] Chow, S. N., Mallet-Paret, J., and Yorke, J. A., 'Finding Zeros of Maps : Homotopy Methods That Are Constructive With Probability One', Mathematics of Computation, volume 32, page 887, 1978.

[7] Draper Research and Development Version of the Goddard Trajectory Determination System (GTDS), Charles Stark Draper Laboratory, Cambridge, Massachusetts.

[8] Long, A. C., Capellari, J. O., Velez, C. E., and Fuchs, A. J., 'Goddard Trajectory Determination System Mathematical Theory, Revision 1', National Aeronautics and Space Administration/Goddard Space Flight Center, July 1989.

[9] Battin, Richard. H., 'An Introduction to the Mathematics and Methods of Astrodynamics', American Institute of Aeronautics and Astronautics Education Series, New York, 1987.

[10] Tangredi, L., GE Astro-Space Division, personal communication to Draper Laboratory.

[11] Williams, Debra. S., 'Landsat 6 : A New Window On Planet Earth', The Space Times, Volume 3, July-August 1988.

[12] Abramowitz, Milton., and Stegun, Irene. A., 'Handbook of Mathematical Functions With Formulas, Graphs and Mathematical Tables', United States Government Printing Office, 1972.

[13] Culp, Robert. D., and Lin-Sheng, Jin., 'Preliminary Orbit Determination Using Satellite-to-Satellite Limited Range and Range-Rate Data', Acta Astronautica, Volume 15, Number 11, Pages 807-811, 1987.

[14] Ball, Joseph. E., 'Preliminary Circular Orbit from a Single Station of Range-Only Data', American Institute of Aeronautics and Astronautics Journal, Volume 5, Number 12, December 1967.

[15] Duong, Nguyen., and Winn, C. Byron., 'Orbit Determination by Range-Only Data', Journal of Spacecraft, Volume 10, Number 2, February, 1972.

[16] McClain, W., 'Landsat 6 Post Ascent Orbit Determination Study', The Charles Stark Draper Laboratory, Prepared for GE Astro-Space Division, January 31, 1990.

APPENDIX A

The purpose of this appendix is to explain the Gaussian quadrature technique used in the arc length correction sub-algorithm. Gaussian Quadrature is a technique used to exactly evaluate the integral of any polynomial of degree $\leq$ (2n-1) using the linear combination of just n evaluations (at specific points). In this appendix, the n = 2 case will be outlined.

Suppose we seek a result of the form :

$$\int_{-1}^{1} f(x)dx = c_1 f(x_1) + c_2 f(x_2) \tag{10.1}$$

We need to find the values of c_1, c_2, x_1 and x_2, such that the equation 10.1 is exact for all polynomials f(x) of degree ≤ 3. By linearity, the expression must hold for all polynomials of degree ≤ 3 if it holds for the monomials x^0, x^1, x^2 and x^3. Thus, our problem is reduced to finding the values of c_1, c_2, x_1 and x_2 such that :

$$\int_{-1}^{1} x^0 dx = 2 = c_1 + c_2$$

$$\int_{-1}^{1} x^1 dx = 0 = c_1 x_1 + c_2 x_2$$

$$\int_{-1}^{1} x^2 dx = \frac{2}{3} = c_1 x_1^2 + c_2 x_2^2$$

$$\int_{-1}^{1} x^3 dx = 0 = c_1 x_1^3 + c_2 x_2^3$$

If we choose $x_1 = -x_2$ and $c_1 = c_2$, then the 2^{nd} and 4^{th} equations are satisfied. The remaining two equations then require $c_1 = 1$ and $x_1 = \pm 1/\sqrt{3}$. Hence, we can say :

$$\int_{-1}^{1} f(x)dx = f(1/\sqrt{3}) + f(-1/\sqrt{3}) \tag{10.2}$$

is true for all polynomials f(x) of degree ≤ 3.

Now we need to be able to generalize equation 10.2 to an integral with general limits. To do this, consider the following :

$$\int_{x_1}^{x_2} f(x)dx$$

Let $x = Au + B$ where A and B are constants. This gives :

$$\int_{x_1}^{x_2} f(x)dx = A \int_{\frac{x_1-B}{A}}^{\frac{x_2-B}{A}} f(Au + B)du \tag{10.3}$$

Now let $\frac{x_1-B}{A} = -1$ and $\frac{x_2-B}{A} = 1$ giving $A = \frac{x_2-x_1}{2}$ and $B = \frac{x_2+x_1}{2}$. Then we can rewrite equation 10.3 as :

$$\int_{x_1}^{x_2} f(x)dx = \frac{x_2 - x_1}{2} \int_{-1}^{1} f\left(\frac{x_2 - x_1}{2}u + \frac{x_2 + x_1}{2}\right) du \tag{10.4}$$

Thus, in accordance with equation 10.2, we can say that for all polynomials f(x) of degree ≤ 3 :

$$\int_{x_1}^{x_2} f(x)dx = \frac{x_2 - x_1}{2}\left[f\left(\frac{x_2 - x_1}{2\sqrt{3}} + \frac{x_2 + x_1}{2}\right) + f\left(-\frac{x_2 - x_1}{2\sqrt{3}} + \frac{x_2 + x_1}{2}\right)\right] \tag{10.5}$$

is an exact equation. This is known as 3^{rd} order, or 2-point Gaussian quadrature. The 3-point or 4^{th} order scheme is used in the arc length evaluation discussed in chapter 3. This will give exact integrals for any polynomial of degree ≤ 4. The Gaussian quadrature coefficients for higher order methods are known and can be found in tabulated form in reference [12]

APPENDIX B

The software consists of the main program, twenty-one subroutines and two block data sub-programs. Two of the subroutines, ludcmp and lubksb, were taken from reference [4]. The two block data sub-programs and nine of the subroutines were taken from the Draper Research and Development version of the Goddard Trajectory Determination System [7]. The mathematical description of these subroutines is contained in [8]. The main program and the remaining ten subroutines were written by the author.

Software layout is as follows :

- newhmtpy — Executive. Contains starter and monitor sub-algorithms explicitly. Performs sequencing of the main steps in the method.
- stpsz — Carries out step size selection.
- predict — Carries out curve point predictor step.
- correct — Carries out curve point corrector step.
- arc — Carries out arc length correction step.
- output — Carries out collection of solution points.
- critical — Carries out collection of critical points.
- terminate — Carries out determination of whether the curve has been completed.
- invert — Carries out matrix inversion.
- tangent — Carries out curve tangent evaluation.

- derive — Carries out determination of partial derivative matrix using numerical differencing.
- brolyd [7] — Carries out orbit propagation.
- crtgeo [7] — Converts geodetic latitude, longitude and height into Earth-fixed Cartesian coordinates.
- ellgeo [7] — Sets size and shape parameters for the Earth.
- cartes [7] — Converts Equinoctial orbit elements to position and velocity in inertial Cartesian coordinates.
- eclong [7] — Solves Kepler's equation for the eccentric longitude using Newton's method for nonlinear equations.
- auxeqn [7] — Computes auxiliary parameters related to the Equinoctial orbit elelments.
- eqnkep [7] — Converts Kepler elements to Equinoctial elements.
- kepeqn [7] — Converts Equinoctial elements to Kepler elements.
- crtdrv [7] — Computes the partial derivatives of current position and velocity with respect to current equinoctial elements.
- ludcmp [4] — Decomposes a square matrix into the product of a lower diagonal matrix and an upper diagonal matrix.
- lubksb [4] — Carries out back substitution method for solving a lower diagonal/upper diagonal system.
- satelm# [7] — Block data sub-program containing information about current orbital elements.
- elipsd# [7] — Block data sub-program containing information the size and shape parameters of the Earth.

Note : Subroutines brolyd, eqnkep and kepeqn have minor modifications to the originals received from Draper.

Software input is provided from the keyboard. Keyboard input may be redirected to a file if file input is desired.

Software writes to the following files :

- solutn.dat — Solution points are stored.
- critcl.dat — Critical points are stored.
- range3.dat — Predicted and corrected points are stored.
- axis.dat — For each corrected point the semi-major axis and the λ coordinate are stored.
- eccent.dat — For each corrected point the eccentricity and the λ coordinate are stored.
- inclin.dat — For each corrected point the inclination and the λ coordinate are stored.
- node.dat — For each corrected point the ascending node and the λ coordinate are stored.
- pergee.dat — For each corrected point the arguement of perigee and the λ coordinate are stored.
- anomly.dat — For each corrected point the true anomaly and the λ coordinate are stored.
- arclen.dat — For each corrected point the curve length s and the λ coordinate are stored.

Data in files 4 to 10 inclusive may be plotted using MATLAB.

Listings of these software are given below :

```
                program newhmtpy
c
c
c  This program computes a satellite epoch state using only six observed
c ranges at six known observation times relative to epoch. The method used is
c the Homotopy Continuation Method. It is based on and modifies the work of
c R. L. Smith and C. Huang.
c
c
c  Required type statements                                              10
c
c
      implicit none (a-z)
      logical setrtr
      double precision mu,long,lat,omega,height,theta,t,temp,satest,x1,
     +y1,z1,xest,eqnelm,pos,vel,yex,yest,stn,drdpv,xex,satex,x,y,sat,
     +deltas,rhs,lhs,drdeq,dpvdeq,theta0,value,comp,comp1,lambda,row1,
     +scale,u,u1,u2,u3,u3old,s,s1,s2,s3,s3old,L1,L2,pi,d,duds,kep,kep1,
     +retro,eps2,duds1,xestin,amat,sign,r8dum,tto,dfrc,bj2,bj3,bj4,bj5,
     +ae,oscele,orbel,sbuffer,buffer,row,iter,relax,a,b,prop,array      20
      integer i,j,k,iflag,jflag,kflag,n,lflag,indx,mflag,ipert,ipass,
     +idmean,intgr,number,nflag,var
c
c
c  Fix the value of n to be 6.
c
c
      parameter(n=6)
c
c  Required common blocks                                                30
c
      common/dcint /r8dum(126),tto,intgr
      common/   frc/ dfrc(1300)
c
c
c  Dimension the various arrays required.
c
c
      dimension xest(n),xex(n),x(n),t(n),eqnelm(n),temp(n),drdpv(3),
     +vel(3),yex(n),yest(n),y(n),stn(3,n),satex(3,n),satest(3,n),pos(3),  40
     +sat(3,n),rhs(7),lhs(7,7),dpvdeq(6,6),drdeq(6,6),indx(7),kep(n),
     +scale(7),u(7),u1(7),u2(7),u3(7),u3old(7),duds(7),kep1(n),iter(3),
     +duds1(7),xestin(n),oscele(n),orbel(5),buffer(7),amat(n),row(7),
     +array(n),row1(n)
c
c  Required equivalence statements
c
      equivalence   (bj2,dfrc(356))     ,(bj3,dfrc(357))     ,
     +              (bj4,dfrc(358))     ,(bj5,dfrc(359))     ,
     +              (dfrc(2),mu)        ,(ae,dfrc(24))                  50

c
c*********************************************************************
c     DEFINITION OF VARIABLES USED IN THIS PROGRAM
```

```
c*************************************************************************
c
c  setrtr: logical variable to determine whether or not to change retro
c  retro: either +1 or -1 depending on whether orbit is retrograde or not
c  (For our purposes retro = +1 and setrtr = .false. at all times)
c  mu: gravitational parameter of the Earth                                  60
c  long: longitude of Earth station in Earth fixed coordinates
c  lat: latitude of Earth station in Earth fixed coordinates
c  omega: Earth's rotational rate
c   height:  local height of Earth station
c   theta:  Greenwich hour angle
c   theta0:  Greenwich hour angle at epoch
c   t:  array holding observation times
c   satest:  array holding satellite position in inertial coordinates
c    corresponding to the a priori epoch estimate
c   satex:  array holding exact satellite position in inertial coordinates at   70
c    the six observation times
c   sat:  array holding satellite position in inertial coordinates corresponding
c    to current epoch estimate
c   x1:  current x-coordinate of station in inertial coordinates
c   y1:  current y-coordinate of station in inertial coordinates
c   z1:  current z-coordinate of station in inertial coordinates
c   xestin:  array holding a priori epoch estimate
c   xest:  array holding epoch estimate at the latest restart point
c   xex:  array holding exact epoch state
c   x:  array holding current epoch estimate                                  80
c   lambda:  a parameter which defines the homotopy (= 0.0 at a priori epoch
c    estimate and = 1.0 at solution)
c   u:  array holding lambda and x
c   u1:          >
c   u2:          > arrays holding lambda and x information at backpoints
c   u3:          > along the solution curve
c   u3old:    >
c   s:  the distance along the solution curve from the start point
c   s1:          >
c   s2:          > distances along the solution curve from the start point at   90
c   s3:          > backpoints along the solution curve
c   s3old:    >
c   L1:      > Lagrange polynomials for two backpoints
c   L2:      >
c   temp:  array holding current elements
c   eqnelm:  array holding current Equinoctial elements
c   pos:  array holding current satellite position in inertial coordinates
c   vel:  array holding current satellite velocity in inertial coordinates
c   y:  array holding ranges corresponding to current epoch estimate
c   yex:  array holding the exact ranges                                      100
c   yest:  array holding ranges corresponding to a priori epoch estimate
c   stn:  array holding station position in inertial coordinates at the six
c    observation times
c   dpvdeq:  array holding partial derivatives of current position and velocity
c    with respect to current Equinoctial elements (this is then multiplied by
c    the partial derivative of current Equinoctial elements with respect to
c    epoch Equinoctial elements and the answer returned back to dpvdeq)
c   drdpv:  array holding partial derivatives of ranges with respect to
```

```
c   current position and velocity
c  drdeq:  array holding partial derivatives of ranges with respect to epoch     110
c   Equinoctial elements
c  deltas:  curve length step
c  iflag:  # of Newton-Raphson iterations
c  jflag:  # of times original deltas has been halved
c  kflag:  # of critical points passed
c  lflag:  # of times program has been restarted
c  mflag:  # of times that the initial start point has been passed
c  nflag:  # of solution points that have been passed
c  rhs:  array holding the right hand side of the 7x7 system to be solved in
c   each Newton-Raphson iteration (the solution is later returned back to rhs)  120
c  lhs:  array holding the left hand side of the 7x7 system to be solved in each
c   Newton-Raphson iteration
c  indx:  array local to the lu decomposition and back substitution routines
c   used for solving the above mentioned 7x7 system
c  d:  variable local to the lu decomposition and back substitution routines
c   used for solving the above mentioned 7x7 system
c  scale:  array holding scaling constants for the u vector
c  eps2:  similar to eps1 but used in a different convergence criterion
c  value:  used in determining convergence ; also used in determining if a
c   time is valid for observation of satellite from station                      130
c  comp:  variable used in determining if a time is valid for observation of
c   satellite from station
c  comp1:  variable used in determining if a time is valid for observation of
c   satellite from station
c  sign:  value of lambda after the very first step along the solution curve
c  duds:    Array holding derivative of the u vector with respect to curve
c   length s at the most recent solution point
c  duds1:  Array holding derivative of the u vector with respect to curve
c   length s at the last back point
c  kep1:    Array holding Kepler elements corresponding to the most recent       140
c   solution point
c  kep:    Array holding Kepler elements corresponding to the last back point
c  number:    A variable whose value is 1 if the program has just emerged from
c   the starter and is 0 otherwise
c  intgr:    A variable which takes the value of 1 if the eccentricity exceeds
c   1 while in subroutine brolyd but is 0 otherwise
c  ipert:      >
c  ipass:      > Variables whose values determine the paths taken in subroutine
c  idmean:    > brolyd
c  tto:    Current time since epoch                                             150
c  bj2:      >
c  bj3:      >  Coefficients of the Earth's gravitational field expansion
c  bj4:  >
c  bj5:  >
c  r8dum:  >  Arrays in the common block with brolyd
c  dfrc:  >
c  ae:  Earth's mean radius in km
c  buffer:    array holding the u vector solution point just before the last
c   restart point
c  sbuffer:    curve length s corresponding to solution point buffer            160
c  oscele:    array holding latest osculating elements
c  iter:    array holding convergence criterion values for the last three
```

```
c   iterations of the N-R scheme
c  relax:    variable holding the value of a relaxation parameter for the N-R
c   scheme
c  prop:    variable whose value determines whether orbit propagation is 2-body
c   mechanics or Brouwer-Lydanne
c  var:    variable whose value determines certain decisions such as if exact
c   ranges should be entered from the keyboard or not
c  array:    array containing either mean elements or osculating elements        170
c   depending on what kind of propagation has been selected
c  amat:      >
c  row:        > arrays used in the calculation of the local tangent vector
c  row1:    array used in the numerical calculation of partial derivatives
c
c*************************************************************************
c
c
c  Open files for output of data.
c                                                                             180
c
      open(1,file = 'range3.dat')
      open(2,file = 'axis.dat')
      open(3,file = 'eccent.dat')
      open(4,file = 'inclin.dat')
      open(7,file = 'node.dat')
      open(8,file = 'pergee.dat')
      open(9,file = 'anomly.dat')
      open(10,file = 'solutn.dat')
      open(11,file = 'critcl.dat')                                            190
      open(13,file = 'arclen.dat')
c
c  Ask user whether he/she would like to use two-body propogation or
c Brouwer-Lydanne propogation
c
      write(6,*)'If you would like two-body propogation enter 1, if you would
     +like Brouwer-Lydanne propogation enter 0'
      read(5,*)prop
c
c                                                                             200
c  Set values of various constants.
c
c
      setrtr = .false.
      retro = 1.0
      omega = 7.2921159e-05
      mu = 398600.64
      eps2 = 1.0d-11
      pi = 3.1415926535897932
      ae = 6.3781350d3                                                        210
      bj2 = -1.082627d-3
      bj3 = 2.536414d-6
      bj4 = 1.6233497d-6
      bj5 = 2.2608567d-7
      kflag = 0
      lflag = 0
```

```
      mflag = 0
      nflag = 0
      ipass = 1
      idmean = 1                                                        220
      if(prop.lt.0.5)then
      ipert = 2
      else
      ipert = 0
      endif
c
c
c  Print message to screen asking for Earth station latitude, longitude
c and height.
c                                                                       230
c
      write(6,*)'Input Earth-fixed station latitude, longitude and heigh
     +t(in km) in that order.  Latitude and longitude in degrees.'
      read(5,*)lat,long,height
      lat = lat*pi/180.0
      long = long*pi/180.0
c
c
c  Calculate the station's Earth fixed cartesian coordinates.
c                                                                       240
c
c
     call crtgeo(x1,y1,z1,height,lat,long)
c
c
c Print message to screen asking for initial angle between inertial and
c Earth-fixed coordinates (Greenwich Hour Angle).
c
c
     write(6,*)'Input the initial angle between inertial and Earth-fix
      +ed coords in degrees.'                                           250
     read(5,*)theta0
     theta0 = theta0*pi/180
c
c
c  Print message to screen asking for observation times in seconds after
c epoch state time, and read them into an array.
c
c
     write(6,*)'Input the six observation times, in seconds, in chronol
      +ogical order.'                                                   260
     read(5,*)(t(i), i = 1,n)
c
c
c Calculate the station's cartesian coordinates in inertial coordinates
c at the six chosen observation times.
c
c
      do 5 i = 1,n
         theta = omega*t(i)+theta0
         stn(1,i) = x1*cos(theta)-y1*sin(theta)                         270
```

```
         stn(2,i) = x1*sin(theta)+y1*cos(theta)
         stn(3,i) = z1
 5       continue
c
c
c  If two body mechanics has been chosen, allow exact ranges to be calculated
c from exact epoch elements or to be entered as observed quantities
c
c
      if(prop.le.0)then                                                    280
      var = 1
      goto 6
      endif
      write(6,*)'If you wish to use observed ranges enter 1, if not, ent
     +er 0'
      read(5,*)var
 6    if(var.gt.0)then
      write(6,*)'Enter observed ranges in kilometers'
      read(5,*)(yex(i), i = 1,n)
      goto 204                                                             290
      endif
c
c
c  Print message to screen asking for a satellite epoch state in Kepler
c elements (Brouwer mean elements), and read them into an array.
c
c
      write(6,*)' Input a satellite epoch state, in Kepler elements, in
     +the following order:   a(km),e,i,w,q,m. All angles in degrees.'
      read(5,*)(xex(i), i = 1,n)                                           300
      do 10 i = 3,6
         xex(i) = xex(i)*pi/180
 10      continue
c
c
c  For each of the six chosen observation times :    1.)  calculate the current
c Kepler elements and put them in an array (temp)  2.)  convert them to
c equinoctial elements using subroutine eqnkep  3.)  convert the equinoctial
c elements to position and velocity in inertial cartesian coordinates
c using subroutine cartes  4.)  write the satellite position to an array     310
c
c
      do 150 i = 1,n
         tto = t(i)
         do 50 j = 1,n
            temp(j) = xex(j)
 50         continue
         call brolyd(oscele,temp,ipert,ipass,idmean,orbel,pi)
         if(prop.lt.0.5)then
         do 55 j = 1,n                                                     320
            array(j) = oscele(j)
 55         continue
         else
         do 60 j = 1,n
```

```
            array(j) = temp(j)
 60         continue
         endif
         call eqnkep(eqnelm,retro,array,setrtr)
         call cartes(pos,vel,eqnelm,retro,mu)
         do 100 j = 1,3                                                    330
            satex(j,i) = pos(j)
 100        continue
 150     continue
c
c
c  Calculate the exact ranges at the six chosen times and store them
c in an array.
c
c
      do 200 i = 1,n                                                       340
         yex(i) = sqrt((satex(1,i)-stn(1,i))**2+(satex(2,i)-stn(2,i))**2
     ++(satex(3,i)-stn(3,i))**2)
 200     continue
c
c
c  Check that at the chosen times, the satellite is in view of the station.
c
c
      do 210 i = 1,n
         value = 0.0                                                       350
         do 205 j = 1,3
            comp = (satex(j,i)-stn(j,i))/yex(i)
            comp1 = stn(j,i)/sqrt(stn(1,i)**2+stn(2,i)**2+stn(3,i)**2)
            value = value+comp*comp1
 205        continue
         if(value.lt.0.175)then
         write(6,*)'Time',t(i),'is a bad time because at that time the s
   +atellite is not in view of the station'
         pause
         endif                                                             360
 210     continue
c
c
c  Write the exact satellite position data to file.
c
c
      write(1,*)'*******************************************************
   +*******'
      write(1,*)'Satellite exact position in inertial coordinates at tim
   +es:'                                                                   370
      write(1,*)' '
      write(1,*)'  t1      t2      t3      t4      t5      t6
   +'
      write(1,*)' '
      do 216 j = 1,3
         write(1,217)(satex(j,i), i = 1,n)
 216     continue
 217  format(6F10.2)
```

```
c
c                                                                      380
c  Write the station coordinates to file.
c
c
 204  write(1,*)'Station position in inertial coordinates at times:'
      write(1,*)' '
      write(1,*)'  t1      t2      t3      t4      t5      t6
     +'
      write(1,*)' '
      do 215 j = 1,3
         write(1,217)(stn(j,i), i = 1,n)                               390
 215     continue
c
c
c  Print message to screen asking for an estimate to the epoch state in
c Kepler elements, and read them into an array.
c
c
      write(6,*)'Input an estimate to the satellite epoch state, in Kepl
     +er elements, in the following order: a(in km),e,i,w,q,m. All angle
     +s in degrees.'                                                   400
      read(5,*)(xest(i), i = 1,n)
      do 218 i = 3,6
         xest(i) = xest(i)*pi/180
 218     continue
c
c  Initialize backpoint information
c
      u1(1) = 0.0
      u2(1) = 0.0
      u3(1) = 0.0                                                      410
      s = 0.0
      s1 = 0.0
      s2 = 0.0
      s3 = 0.0
c
c Save a priori Kepler elements in array kep
c
      do 219 i = 1,n
         kep(i) = xest(i)
 219     continue                                                      420
c
c  Convert a priori estimate to Equinoctial elements
c
      call eqnkep(xest,retro,xest,setrtr)
c
c Write first point data to plotting files
c
      write(2,'(1x,2F25.16)')u1(1),kep(1)
      write(3,'(1x,2F25.16)')u1(1),kep(2)
      write(4,'(1x,2F25.16)')u1(1),kep(3)*180.0/pi                     430
      write(7,'(1x,2F25.16)')u1(1),kep(4)*180.0/pi
      write(8,'(1x,2F25.16)')u1(1),kep(5)*180.0/pi
```

```
      write(9,'(1x,2F25.16)')u1(1),kep(6)*180.0/pi
      write(13,'(1x,2F25.16)')u1(1),s1
c
c  Initialize the backpoint information
c
      do 220 i = 1,n
         u1(i+1) = xest(i)
         u2(i+1) = xest(i)                                              440
         u3(i+1) = xest(i)
         xestin(i) = xest(i)
 220     continue
c
c
c  For each of the chosen observation times calculate the present Kepler
c elements and convert to present Equinoctial elements
c
c
      do 260 i = 1,n                                                    450
         tto = t(i)
         do 250 j = 1,n
            temp(j) = kep(j)
 250        continue
         call brolyd(oscele,temp,ipert,ipass,idmean,orbel,pi)
         if(prop.lt.0.5)then
         do 251 j = 1,n
            array(j) = oscele(j)
 251        continue
         else                                                           460
         do 252 j = 1,n
            array(j) = temp(j)
 252        continue
         endif
         call eqnkep(eqnelm,retro,array,setrtr)
c
c  Convert from Equinoctial elements to position and velocity using
c subroutine cartes.  This subroutine also sets up the satelm block data
c to correspond to the current time.
c                                                                       470
         call cartes(pos,vel,eqnelm,retro,mu)
c
c  Write the estimated satellite position to an appropriate array.
c
         do 255 j = 1,3
            satest(j,i) = pos(j)
 255        continue
c
c  Calculate the estimated range and store it in the appropriate array.
c                                                                       480
         yest(i) = sqrt((satest(1,i)-stn(1,i))**2+(satest(2,i)-stn(2,i))
     +**2+(satest(3,i)-stn(3,i))**2)
 260     continue
c
c
c  Write the actual and estimated range data to file.
```

```
c
c
      write(1,*)'*****************************************************
     +*******'                                                          490
      write(1,*)' '
      write(1,*)'          RANGES'
      write(1,*)' '
      write(1,*)'     exact        estimated  '
      write(1,*)' '
      do 270 i = 1,n
         write(1,*)yex(i),yest(i)
 270     continue
      write(1,*)' '
c                                                                       500
c calculate amat array
c
      do 267 i = 1,n
         amat(i) = yex(i)-yest(i)
 267     continue
c
c
c  Bootstrap starter begins here!  Input an initial value for deltas.
c
c                                                                       510
      lambda = 0.0
c
c  Enter +1 or -1 to determine which direction you want to start the curve in
c
      write(6,*)'Enter +1 or -1 to determine which direction you want to
     + start the curve in'
      read(5,*)deltas
      deltas = 0.1*deltas
      goto 272
c                                                                       520
c  If the program reaches this point, we are restarting at the initial point
c or terminating the program
c
 271  write(6,*)'Either input a deltas that is greater than 10 which wil
     +l terminate the program, or input a different deltas, such as the
     +negative of the original deltas used, and the program will restart
     + at the original start point'
      read(5,*)deltas
      deltas = 0.1*deltas
      mflag = 1                                                         530
      do 265 i = 1,n
         xest(i) = xestin(i)
 265     continue
      s = 0.0
      s1 = 0.0
      s2 = 0.0
      s3 = 0.0
      lambda = 0.0
      u2(1) = 0.0
      u3(1) = 0.0                                                       540
```

```
      do 266 i = 1,n
         u2(i+1) = xest(i)
         u3(i+1) = xest(i)
 266     continue
      lflag = 0
      kflag = 0
 272  if(deltas.ge.1)then
      goto 1000
      endif
c                                                                       550
c  Set up scaling for u components, i.e., for the Equinoctial elements
c
      scale(1) = 1.0
      scale(2) = 1000
      scale(3) = 0.001
      scale(4) = 0.001
      scale(5) = 1.0
      scale(6) = 1.0
      scale(7) = 1.0
c                                                                       560
c  Set counter for number of times deltas is halved to zero.
c
      jflag = 0
      goto 280
 275  deltas = deltas/2.0
c
c  Initiate loop which adds deltas to each of the seven dimensions of u in turn
c
 280  do 450 i = 1,7
         s1 = s1+abs(deltas*scale(i))                                   570
c
c  Initialize iter array
c
         iter(1) = 100.0
         iter(2) = 100.0
         iter(3) = 100.0
c
c  Set Newton-Raphson iteration counter to 0 and relaxation parameter to 1
c initially
c                                                                       580
         iflag = 0
         relax = 1.0
c
c Set up u1 = (lambda,xest) initially
c
         do 290 j = 1,n
            u1(j+1) = xest(j)
 290        continue
         u1(1) = lambda
c                                                                       590
c  Add deltas to the relevant entry in u1
c
         u1(i) = u1(i)+deltas*scale(i)
c
```

```
c  Set up x = u1(2) to u1(7)
c
         do 300 j = 1,n
            x(j) = u1(j+1)
 300        continue
c                                                                      600
c  Calculate the Jacobian matrix and range vector corresponding to this x
c vector
c
c  For each of the chosen observation times calculate the current Kepler
c elements and convert to current Equinoctial elements
c
 310     do 360 j = 1,n
            tto = t(j)
            call kepeqn(temp,x,retro,kep)
            call brolyd(oscele,temp,ipert,ipass,idmean,orbel,pi)       610
            if(intgr.eq.1)then
            goto 450
            endif
            if(prop.lt.0.5)then
            do 311 k = 1,n
               array(k) = oscele(k)
 311           continue
            else
            do 312 k = 1,n
               array(k) = temp(k)                                      620
 312           continue
            endif
            call eqnkep(eqnelm,retro,array,setrtr)
c
c  Convert from Equinoctial elements to position and velocity using
c subroutine cartes.  This subroutine also sets up the satelm block data
c to correspond to the current time.
c
            call cartes(pos,vel,eqnelm,retro,mu)
c                                                                      630
c  Write the current satellite position to an appropriate array.
c
            do 355 k = 1,3
               sat(k,j) = pos(k)
 355           continue
c
c  Calculate the current range and store it in the appropriate array.
c
            y(j) = sqrt((sat(1,j)-stn(1,j))**2+(sat(2,j)-stn(2,j))**
     +2+(sat(3,j)-stn(3,j))**2)                                        640
c
c  Calculate derivative matrix either analytically in the case of 2-body
c mechanics or by numerical differencing in the case of Brouwer propagation
c
            if(prop.lt.0.5)then
c
c  Calculate the matrix of partial derivatives of ranges with respect to epoch
c mean Brouwer elements using subroutine derive
```

```
c
            call derive(row1,y(j),x,retro,kep,ipass,ipert,idmean,
     +pi,setrtr,mu,j,stn)
            if(intgr.eq.1)then
            goto 450
            endif
            do 359 k = 1,n
               drdeq(j,k) = row1(k)
 359           continue
            goto 360
            else
c
c  Calculate the matrix of partial derivatives of current position and
c velocity with respect to current equinoctial elements using subroutine
c crtdrv.
c
            call crtdrv(dpvdeq)
c
c  Multiply the matrix of partial derivatives(dpvdeq) by the matrix of
c partial derivatives of current state in Equinoctial elements with respect to
c epoch state in Equinoctial elements, and return the resulting matrix back
c to dpvdeq.
c   Note that since the matrix of partial derivatives of current state in
c Equinoctial elements, with respect to epoch state in Equinoctial elements,
c is the six-by-six identity except that the (6,1) entry is :
c -1.5*sqrt(mu/a**5)*t.
c
            do 356 k = 1,n
               dpvdeq(k,1) = dpvdeq(k,1)-1.5*sqrt(mu/x(1)**5)*t(j)*d
     +pvdeq(k,6)
 356           continue
c
c  Calculate the row vector (drdpv) resulting from partial differentiation of
c range at the current time with respect to current position and velocity.  The
c last three entries are zeroes since range is independent of current velocity
c components.
c
            do 357 k = 1,3
               drdpv(k) = (sat(k,j)-stn(k,j))/y(j)
 357           continue
c
c  Carry out the multiplication drdpv x dpvdeq to get the ith row of the
c Jacobian matrix (drdeq).
c
            do 358 k = 1,n
               drdeq(j,k) = drdpv(1)*dpvdeq(1,k)+drdpv(2)*dpvdeq(2,k
     +)+drdpv(3)*dpvdeq(3,k)
 358           continue
            endif
 360        continue
c
c  Set up the 7x7 system to be solved in order to get the change in u1, deltau
c
c  Calculate the 7x1 right hand side column vector, rhs.  Also set up a
```

```
c convergence criterion on rhs
c
         value = 0.0
         do 370 j = 1,n
            rhs(j+1) = -yest(j)+y(j)-u1(1)*amat(j)
            value = max(value,abs(rhs(j+1)/max(abs(yex(j)),ab
     +s(yest(j)),abs(y(j)))))
 370        continue                                                    710
         write(1,*)value,u1(1)
         if(value.le.eps2)then
         deltas = abs(deltas*scale(i))
         goto 460
         endif
c
c  Update iteration array
c
         iter(3) = iter(2)
         iter(2) = iter(1)                                           720
         iter(1) = value
c
c  If convergence criterion is not met, update iteration number and if it is
c less than 10, update u1 and do the next iteration.  If the iteration number is
c not less than 10 then return to the start of the bootstrap starter and add
c deltas to the next component of the initial u1.
c
         iflag = iflag+1
         if(iflag.gt.10)then
         goto 450                                                    730
         endif
         rhs(1) = 0.0
c
c  Calculate the 7x7 left hand side matrix, lhs
c
         do 390 j = 1,n
            lhs(j+1,1) = amat(j)
            do 380 k = 1,n
               lhs(j+1,k+1) = -drdeq(j,k)
 380           continue                                                740
 390        continue
         call tangent(amat,drdeq,row)
         do 400 j = 1,7
            lhs(1,j) = row(j)
            duds1(j) = lhs(1,j)
 400        continue
c
c  Solve the 7x7 system using subroutines ludcmp and lubksb which return deltau
c as rhs
c                                                                    750
         call ludcmp(lhs,7,7,indx,d)
         call lubksb(lhs,7,7,indx,rhs)
c
c  If the scheme has not converged but appears to be doing so, then
c allow another iteration with current value of relax.  If it was converging
c but the last iteration was a divergence then set relax to 0.8 and do the
```

```
c next iteration.  If it is diverging try the next component of u.
c
         if(iter(1).le.iter(2))then
         goto 415                                                        760
         elseif(iter(1).gt.iter(2).and.iter(2).le.iter(3))then
         relax = 0.8
         goto 415
         elseif(iter(1).gt.iter(2).and.iter(2).gt.iter(3))then
         goto 450
         endif
c
c  Add the calculated change
c
 415     do 420 j = 1,n                                                  770
            x(j) = x(j)+relax*rhs(j+1)
 420        continue
         u1(1) = u1(1)+relax*rhs(1)
c
c  Update u1
c
         do 430 j = 1,n
            u1(j+1) = x(j)
 430        continue
c                                                                        780
c  If eccentricity is greater than 1, try next component of u1
c
         if((x(2)**2+x(3)**2).ge.1.0)then
         goto 450
         endif
         goto 310
 450     continue
c
c  Update the number of times deltas has been halved
c                                                                        790
         jflag = jflag+1
c
c  If you reach this point that means that the bootstrap starter has not
c worked with the current value of deltas.  If the number of times that the
c deltas has been halved is less than 10, then halve it again and start once
c more.  If the deltas has been halved not less than 10 times, consider the
c starter as having failed.
c
      if(jflag.lt.10)then
      goto 275                                                           800
      endif
c
c  If bootstrap starter failed, write failure message to screen
c
      write(6,*)'Starter failed. To terminate the program type 0, to res
   +tart at the initial start point type 1'
      read(5,*)var
      if(var.gt.0.5)then
      goto 271
      endif                                                              810
```

```
      goto 1000
c
c Convert to Kepler elements for output
c
 460  call kepeqn(kep,x,retro,kep)
c
c
c  Write important information to file
c
c                                                                      820
      write(1,*)' '
      write(1,*)'Value of deltas that gave convergence is',deltas,'.'
      write(1,*)' '
      write(1,*)'The corrected next point on the solution curve is:'
      write(1,*)'                lambda =',u1(1),' '
      write(1,*)'                   a =',kep(1),'km'
      write(1,*)'                   e =',kep(2),' '
      write(1,*)'                   i =',kep(3),'radians'
      write(1,*)'                   w =',kep(4),'radians'
      write(1,*)'                   W =',kep(5),'radians'                830
      write(1,*)'                   m =',kep(6),'radians'
c
c
c  Correct arc length
c
c
      value = 0.0
      do 465 i = 1,7
         value = value+(u1(i)-u2(i))**2
 465     continue                                                      840
      s1 = s2+sqrt(value)
c
c  Write data to plotting files and screen
c
      write(2,'(1x,2F25.16)')u1(1),kep(1)
      write(3,'(1x,2F25.16)')u1(1),kep(2)
      write(4,'(1x,2F25.16)')u1(1),kep(3)*180.0/pi
      write(7,'(1x,2F25.16)')u1(1),kep(4)*180.0/pi
      write(8,'(1x,2F25.16)')u1(1),kep(5)*180.0/pi
      write(9,'(1x,2F25.16)')u1(1),kep(6)*180.0/pi                     850
      write(13,'(1x,2F25.16)')u1(1),s1
      write(6,*)u1(1)
c
c  Update number of times program has been restarted
c
      lflag = lflag+1
      if(lflag.eq.1)then
      sign = u1(1)
      endif
      number = 1                                                       860
      if(lflag.gt.1)then
c
c  Test for output states
c
```

```
      if(abs(1.0-u1(1)).gt.0.1.and.abs(1.0-u2(1)).gt.0.1)then
      goto 700
      endif
      call output(u1,u2,u3,s1,s2,s3,stn,yest,yex,mu,retro,setrtr,t,
     +scale,kep,comp,ipass,ipert,idmean,pi,orbel,oscele,number,eps2,
     +nflag,amat,prop)                                                  870
      if(comp.gt.5)then
      write(6,*)'Program terminated because corrector failed to converge
    + within output subroutine immediately after coming out of a
    +restart'
      goto 1000
      endif
c
c  If maximum number of solution points has been reached terminate program
c
      if(nflag.ge.10)then                                               880
      write(6,*)'Maximum number of solution points has been reached. Pro
    +gram terminates'
      goto 1000
      endif
c
c  Check for critical point
c
 700  if(u2(1).gt.u1(1).and.u3(1).gt.u2(1))then
      goto 710
      endif                                                             890
      if(u1(1).gt.u2(1).and.u2(1).gt.u3(1))then
      goto 710
      endif
      call critical(u1,u2,buffer,s1,s2,sbuffer,kep,pi,kflag,stn,yest,
     +yex,mu,retro,setrtr,t,scale,comp,ipass,ipert,idmean,orbel,oscele,
     +amat,prop,eps2)
      if(kflag.ge.10)then
      write(6,*)'Number of critical points is maximum allowed'
      goto 271
      endif                                                             900
c
c  Check for termination condition
c
 710  if(abs(u1(1)).gt.0.1.and.abs(u2(1)).gt.0.1)then
      goto 720
      endif
      call terminate(u1,u2,u3,s1,s2,s3,stn,yest,yex,mu,retro,setrtr,t,
     +xestin,comp,scale,kep,eps2,sign,mflag,ipass,ipert,idmean,pi,orbel,
     +oscele,number,amat,prop)
      if(comp.gt.15)then                                                910
      goto 1000
      endif
      if(comp.gt.5)then
      write(6,*)'Program terminated because corrector failed to converge
    + within terminate subroutine immediately after coming out of a
    +restart'
      goto 1000
      endif
```

```
      endif
 720  number = 0                                                        920
c
c  Select next step size
c
      call stpsz(iflag,deltas)
 470  s = s1+deltas
c
c  Predict the next step along the solution curve
c
      L1 = (s-s2)/(s1-s2)
      L2 = (s-s1)/(s2-s1)                                               930
      do 480 i = 1,7
         u(i) = L1*u1(i)+L2*u2(i)
 480     continue
c
c  Update back point information
c
 489     do 490 i = 1,7
         u3old(i) = u3(i)
         u3(i) = u2(i)
         u2(i) = u1(i)                                                  940
         u1(i) = u(i)
 490     continue
      do 491 i = 1,n
         x(i) = u1(i+1)
 491     continue
      s3old = s3
      s3 = s2
      s2 = s1
      s1 = s
c                                                                       950
c Convert to Kepler elements for output
c
      call kepeqn(kep1,x,retro,kep)
c
c  Write the predicted next point data to file
c
      write(1,*)' '
      write(1,*)'The predicted next point on the solution curve is:'
      write(1,*)'              lambda =',u1(1),' '
      write(1,*)'                 a =',kep1(1),'km'                     960
      write(1,*)'                 e =',kep1(2),' '
      write(1,*)'                 i =',kep1(3),'radians'
      write(1,*)'                 w =',kep1(4),'radians'
      write(1,*)'                 W =',kep1(5),'radians'
      write(1,*)'                 m =',kep1(6),'radians'
c
c  If stepsize is less than 1.0d-5, allow restart options at either the last
c corrected point or at the initial start point or allow terminate option
c
      if(deltas.lt.1.0d-5)then                                          970
      write(6,*)'For a restart at the last corrected point type 1, for a
     + restart at the initial start point type 2, to terminate type 0'
```

```
      read(5,*)var
      if(var.gt.0.5.and.var.lt.1.5)then
      lambda = u2(1)
      do 499 i = 1,n
         xest(i) = u2(i+1)
 499     continue
      do 500 i = 1,7
         buffer(i) = u3(i)                                          980
         u1(i) = u2(i)
         u3(i) = u2(i)
         u3old(i) = u2(i)
 500     continue
      sbuffer = s3
      s1 = s2
      s3 = s2
      s3old = s2
      deltas = 0.1
      goto 272                                                      990
      elseif(var.gt.1.5.and.var.lt.2.5)then
      goto 271
      else
      goto 1000
      endif
      endif
c
c
c  Correct the next step along the solution curve
c                                                                  1000
c
      call correct(u1,stn,yest,yex,mu,retro,setrtr,t,iflag,comp,eps2,
     +duds,kep,ipass,ipert,idmean,pi,orbel,oscele,amat,prop)
c
c  Calculate the dot product of the tangent vectors at the last two
c corrected curve points
c
      a = 0
      b = 0
      value = 0                                                    1010
      do 492 i = 1,7
         value = value+duds(i)*duds1(i)
         a = a+duds(i)**2
         b = b+duds1(i)**2
 492     continue
      value = value/sqrt(a*b)
      write(1,*)' '
      write(1,*)'value =',value,' '
      write(1,*)' '
c                                                                  1020
c  Update x
c
 493  do 495 i = 1,n
         x(i) = u1(i+1)
 495     continue
c
```

```
c  If:
c     1.)  the corrector did not converge
c     2.)  the change in lambda between the last 2 corrected points > 0.1
c     3.)  the change in curve tangent is too great                          1030
c
c  halve deltas, set back the backpoint data  and return to the predictor step
c
 496  if(comp.gt.5.or.abs(u1(1)-u2(1)).gt.0.1.or.abs(value).lt.0.9)then
      deltas = deltas/2.0
      do 497 i = 1,7
         u1(i) = u2(i)
         u2(i) = u3(i)
         u3(i) = u3old(i)
 497     continue                                                           1040
      s1 = s2
      s2 = s3
      s3 = s3old
      if(abs(s2-s3).lt.1.0d-10)then
      goto 470
      endif
      goto 560
      endif
c
c Convert to Kepler elements for output                                     1050
c
      call kepeqn(kep1,x,retro,kep)
c
c  Otherwise, write the corrected next point data to files
c
      write(1,*)' '
      write(1,*)'The corrected next point on the solution curve is:'
      write(1,*)'              lambda =',u1(1),' '
      write(1,*)'                   a =',kep1(1),'km'
      write(1,*)'                   e =',kep1(2),' '                      1060
      write(1,*)'                   i =',kep1(3),'radians'
      write(1,*)'                   w =',kep1(4),'radians'
      write(1,*)'                   W =',kep1(5),'radians'
      write(1,*)'                   m =',kep1(6),'radians'
c
c
c  Correct the arc length
c
c
      call arc(s1,s2,s3,u1,u2,u3)                                           1070
      write(2,'(1x,2F25.16)')u1(1),kep1(1)
      write(3,'(1x,2F25.16)')u1(1),kep1(2)
      write(4,'(1x,2F25.16)')u1(1),kep1(3)*180.0/pi
      write(7,'(1x,2F25.16)')u1(1),kep1(4)*180.0/pi
      write(8,'(1x,2F25.16)')u1(1),kep1(5)*180.0/pi
      write(9,'(1x,2F25.16)')u1(1),kep1(6)*180.0/pi
      write(13,'(1x,2F25.16)')u1(1),s1
      write(6,*)u1(1)
c
c  Update kep                                                               1080
```

```
c
      do 550 i = 1,6
         kep(i) = kep1(i)
 550     continue
c
c  Update duds
c
      do 555 i = 1,7
         duds1(i) = duds(i)
 555     continue                                                    1090
c
c  Calculate the straight line distance between the last corrected point and
c the start point.  If this is greater than the corrected arc length there is
c clearly something wrong
c
      value = 0.0
      do 556 i = 1,7
         value = value+(u1(i)-u2(i))**2
 556     continue
      value = sqrt(value)                                            1100
      if(value.ge.abs(s1))then
      write(6,*)'Corrected arc length is less than physically possible'
      pause
      endif
c
c
c  Collect output states
c
c
      if(abs(1.0-u1(1)).gt.0.1.and.abs(1.0-u2(1)).gt.0.1)then        1110
      goto 590
      endif
      call output(u1,u2,u3,s1,s2,s3,stn,yest,yex,mu,retro,setrtr,t,scale
     +,kep,comp,ipass,ipert,idmean,pi,orbel,oscele,number,eps2,nflag,
     +amat,prop)
c
c  If, while in subroutine output, subroutine correct is called but the N-R
c scheme does not converge, then halve deltas, set back the backpoint
c data and return to the predictor step
c                                                                    1120
      if(comp.gt.5)then
      goto 496
      endif
c
c  If maximum number of solution points has been reached terminate program
c
      if(nflag.ge.10)then
      write(6,*)'Maximum number of solution points has been reached. Pro
   +gram terminates'
      goto 1000                                                      1130
      endif
c
c  Collect critical points
c
```

```
 590  if(u2(1).gt.u1(1).and.u3(1).gt.u2(1))then
      goto 600
      endif
      if(u1(1).gt.u2(1).and.u2(1).gt.u3(1))then
      goto 600
      endif                                                          1140
      call critical(u1,u2,u3,s1,s2,s3,kep,pi,kflag,stn,yest,yex,mu,retro
     +,setrtr,t,scale,comp,ipass,ipert,idmean,orbel,oscele,amat,prop,
     +eps2)
c
c  If maximum number of critical points has been reached allow the option of
c restarting at the initial start point or of terminating
c
      if(kflag.ge.10)then
      write(6,*)'Number of critical points is maximum allowed'
      goto 271                                                       1150
      endif
c
c
c  Check for return to initial state.  If returned, terminate algorithm, if not,
c do next step.
c
c
 600  if(abs(u1(1)).gt.0.1.and.abs(u2(1)).gt.0.1)then
      goto 650
      endif                                                          1160
      call terminate(u1,u2,u3,s1,s2,s3,stn,yest,yex,mu,retro,setrtr,t,
     +xestin,comp,scale,kep,eps2,sign,mflag,ipass,ipert,idmean,pi,
     +orbel,oscele,number,amat,prop)
      if(comp.gt.15)then
      goto 1000
      endif
c
c  If, while in subroutine terminate, subroutine correct is called but the N-R
c scheme does not converge, then halve deltas, set back the backpoint
c data and return to the predictor step                              1170
c
      if(comp.gt.5)then
      goto 496
      endif
c
c  Calculate next step-size
c
 650  call stpsz(iflag,deltas)
 560  s = s1+deltas
c                                                                    1180
c  Predict next step
c
      call predict(s,s1,s2,s3,u,u1,u2,u3)
c
c  If predicted point has a lambda value that exceeds the lambda value of
c the last corrected point by more than 0.1, halve the step size and make a
c new prediction
c
```

```
      if(abs(u(1)-u1(1)).gt.0.1)then
      deltas = deltas/2.0                                              1190
      goto 560
      endif
      goto 489
c
c
c  Close files and end.
c
c
 1000 close(1)
      close(2)                                                         1200
      close(3)
      close(4)
      close(7)
      close(8)
      close(9)
      close(10)
      close(11)
      close(13)
      end
```

```
      subroutine output(u1,u2,u3,s1,s2,s3,stn,yest,yex,mu,retro,setrtr,
     +t,scale,kep,comp,ipass,ipert,idmean,pi,orbel,oscele,number,eps2,
     +nflag,amat,prop)
c
c
c  This subroutine collects any output states that occur between the last
c two points calculated on the solution curve
c
c  Required type statements
c                                                                      10
      implicit none (a-z)
      logical setrtr
      double precision u1,u2,u3,uint,s1,s2,s3,sint,stn,uup,kep1,pi,eps2,
     +udown,yest,yex,mu,t,ua,u,scale,x,kep,retro,comp,duds,var,
     +pi,orbel,oscele,L1,L2,value,soltn,tto,eqnelm,pos,vel,r8dum,amat,
     +prop,array
      integer i,j,iflag,n,ipass,ipert,idmean,number,nflag,k,intgr
c
c  Set n = 6
c                                                                      20
      parameter(n=6)
c
c  Required common blocks
c
      common/dcint /r8dum(126),tto,intgr
c
c  Required dimension statements
c
      dimension u1(7),u2(7),u3(7),uint(7,0:50),sint(0:49),scale(7),
     +stn(3,n),yest(n),yex(n),t(n),uup(7),udown(7),ua(7),u(7),x(n),      30
     +kep(n),kep1(n),duds(7),orbel(5),oscele(n),soltn(n,0:9),pos(3),
     +vel(3),eqnelm(n),amat(n),array(n)
      comp = 0
c
c  Let uint(1:7,0) = u2(1:7), uint(1:7,50) = u1(1:7), and sint(0) = s2
c
      do 25 i = 1,7
        uint(i,0) = u2(i)
        uint(i,50) = u1(i)
 25     continue                                                       40
      sint(0) = s2
c
c  Otherwise, calculate 49 points on the solution curve evenly spaced between
c s1 and s2 using the predictor method ,based on the last two points if we
c have just come out of the starter or based on the last three points otherwise
c
      if(number.eq.1)then
      do 40 i = 1,49
        sint(i) = s2+i*(s1-s2)/50.0
        L1 = (sint(i)-s2)/(s1-s2)                                     50
        L2 = (sint(i)-s1)/(s2-s1)
        do 30 j = 1,7
          uint(j,i) = L1*u1(j)+L2*u2(j)
```

```
 30        continue
 40      continue
     goto 80
     endif
     do 75 i = 1,49
       sint(i) = s2+i*(s1-s2)/50.0
       call predict(sint(i),s1,s2,s3,u,u1,u2,u3)                    60
       do 50 j = 1,7
         uint(j,i) = u(j)
 50       continue
 75     continue
c
c
c  Pick out pairs of these predicted solutions that straddle the lambda = 1
c hyperplane.
c
c                                                                   70
 80   do 200 i = 1,50
       if((1.0-uint(1,i))*(1.0-uint(1,i-1)).ge.0)then
       goto 200
       endif
c
c  For each such pair, correct them using N-R. If corrector does not work, set
c comp = 10 and return to main program
c
       do 125 j = 1,7
         uup(j) = uint(j,i)                                         80
         udown(j) = uint(j,i-1)
 125       continue
       if(i.eq.1)then
       goto 130
       endif
       call correct(udown,stn,yest,yex,mu,retro,setrtr,t,iflag,comp,
    +eps2,duds,kep,ipass,ipert,idmean,pi,orbel,oscele,amat,prop)
       if(comp.gt.5)then
       goto 300
       endif                                                        90
       if(i.eq.50)then
       goto 140
       endif
 130     call correct(uup,stn,yest,yex,mu,retro,setrtr,t,iflag,comp,
    +eps2,duds,kep,ipass,ipert,idmean,pi,orbel,oscele,amat,prop)
       if(comp.gt.5)then
       goto 300
       endif
c
c  Let ua be a linearly interpolated state at lambda = 1             100
c
 140     do 150 j = 1,7
         ua(j) = udown(j)+(1.0-udown(1))*(uup(j)-udown(j))/(uup(1)-ud
    +own(1))
 150       continue
c
c  Correct ua using N-R
```

```
c
      call correct(ua,stn,yest,yex,mu,retro,setrtr,t,iflag,comp,eps2,
     +duds,kep,ipass,ipert,idmean,pi,orbel,oscele,amat,prop)                110
      if(comp.gt.5)then
      goto 300
      endif
c
c  If the corresponding lambda is equal to 1 within a specified tolerance,
c write solution to file unless the same solution has been written to file
c previously. If not, let either uup or udown, whichever is further
c from 1, equal ua, and repeat
c
      if(abs(1.0-ua(1)).gt.1.0d-8)then                                       120
      if(abs(uup(1)-1.0).gt.abs(udown(1)-1.0))then
      do 175 j = 1,7
        uup(j) = ua(j)
175     continue
      goto 140
      endif
      do 190 j = 1,7
        udown(j) = ua(j)
190     continue
      goto 140                                                               130
      endif
c
c  Compare this solution with previous solutions stored to see if it is
c different
c
      do 88 k = 0,nflag
        value = 0.0
        do 85 j = 1,n
          if(j.le.5)then
          value = value+((soltn(j,k)-ua(j+1))/scale(j+1))**2                 140
          goto 85
          endif
          var = abs(mod(soltn(j,k),2*pi)-mod(ua(j+1),2*pi))
          if(var.gt.pi)then
          var = var-2*pi
          endif
          value = value+(var/scale(j+1))**2
85        continue
        if(value.lt.1.0d-2)then
        write(10,*)' '                                                       150
        write(10,*)'A solution point has been encountered but it is
     +the same as one which has been recorded before'
        write(10,*)' '
        goto 300
        endif
88      continue
c
c  Store new solution point in soltn and update number of solutions
c
      do 195 j = 1,n                                                         160
        x(j) = ua(j+1)
```

```
        soltn(j,nflag) = ua(j+1)
 195      continue
       nflag = nflag+1
c
c  Convert to Kepler elements for output
c
       call kepeqn(kep1,x,retro,kep)
c
c Write Brouwer mean solution to file                                    170
c
       write(10,*)' '
       write(10,*)'One solution state in Brouwer mean elements is:'
       write(10,*)'                          a =',kep1(1),'km'
       write(10,*)'                          e =',kep1(2),' '
       write(10,*)'                          i =',kep1(3)*180/pi,'degrees'
       write(10,*)'                          w =',kep1(4)*180/pi,'degrees'
       write(10,*)'                          q =',kep1(5)*180/pi,'degrees'
       write(10,*)'                          m =',kep1(6)*180/pi,'degrees'
       write(10,*)' '                                                    180
c
c  Convert to osculating elements
c
       tto = 0
       call brolyd(oscele,kep1,ipert,ipass,idmean,orbel,pi)
c
c  If Brouwer-Lydanne propagation was being used write the osculating
c solution to file
c
       if(prop.lt.0.5)then                                               190
       write(10,*)' '
       write(10,*)'That solution state in Brouwer osculating elements i
      +s:'
       write(10,*)'                        a =',oscele(1),'km'
       write(10,*)'                        e =',oscele(2),' '
       write(10,*)'                        i =',oscele(3)*180/pi,'degrees'
       write(10,*)'                        w =',oscele(4)*180/pi,'degrees'
       write(10,*)'                        q =',oscele(5)*180/pi,'degrees'
       write(10,*)'                        m =',oscele(6)*180/pi,'degrees'
       write(10,*)' '                                                    200
       do 55 j = 1,n
         array(j) = oscele(j)
 55      continue
       else
       do 60 j = 1,n
         array(j) = kep1(j)
 60      continue
       endif
c
c  Convert to position and velocity                                      210
c
       call eqnkep(eqnelm,retro,array,setrtr)
       call cartes(pos,vel,eqnelm,retro,mu)
c
c  Write solution in terms of position and velocity to file
```

```
c
      write(10,*)' '
      write(10,*)'That solution state in position and velocity is:'
      write(10,*)'                          x =',pos(1),'km'
      write(10,*)'                          y =',pos(2),'km'                    220
      write(10,*)'                          z =',pos(3),'km'
      write(10,*)'                       xdot =',vel(1),'km/s'
      write(10,*)'                       ydot =',vel(2),'km/s'
      write(10,*)'                       zdot =',vel(3),'km/s'
      write(10,*)' '
 200    continue
     comp = 0
 300  end
```

```
      subroutine predict(s,s1,s2,s3,u,u1,u2,u3)
c
c  This subroutine predicts a point along the solution curve using
c quadratic Lagrange extrapolation or interpolation
c
c
c  Required type statements
c
      implicit none (a-z)
      double precision u,u1,u2,u3,s,s1,s2,s3,L1,L2,L3                    10
      integer i
c
c  Dimension the various arrays used
c
      dimension u(7),u1(7),u2(7),u3(7)
c
c  Calculate the Lagrange coefficients
c
      L1 = ((s-s2)*(s-s3))/((s1-s2)*(s1-s3))
      L2 = ((s-s1)*(s-s3))/((s2-s1)*(s2-s3))                             20
      L3 = ((s-s1)*(s-s2))/((s3-s1)*(s3-s2))
c
c  Calculate predicted point u
c
      do 480 i = 1,7
        u(i) = L1*u1(i)+L2*u2(i)+L3*u3(i)
 480    continue
      end
```

```
      subroutine correct(u1,stn,yest,yex,mu,retro,setrtr,t,iflag,comp,
     +eps2,duds,kep,ipass,ipert,idmean,pi,orbel,oscele,amat,prop)
c
c
c  This subroutine takes a predicted solution point and corrects to a real
c solution point using the Newton-Raphson scheme
c
c
c  Required type statements
c                                                                       10
      implicit none (a-z)
      double precision u1,x,temp,mu,t,sat,pos,vel,y,stn,yest,yex,lhs,
     +rhs,value,amat,iter,dpvdeq,drdeq,drdpv,retro,d,comp,eps2,duds,kep,
     +eqnelm,pi,orbel,oscele,r8dum,tto,row,relax,prop,array,row1
      logical setrtr
      integer iflag,j,n,k,indx,ipass,idmean,ipert,intgr
c
c  Fix n to be 6
c
      parameter(n=6)                                                    20
c
c  Required common blocks
c
      common/dcint /r8dum(126),tto,intgr
c
c  Dimension local arrays
c
      dimension u1(7),x(n),temp(n),t(6),sat(3,n),pos(3),vel(3),
     +y(6),stn(3,n),yest(n),yex(n),lhs(7,7),rhs(7),indx(7),dpvdeq(n,n),
     +drdeq(n,n),drdpv(3),duds(7),kep(n),eqnelm(n),orbel(5),oscele(n),   30
     +row(7),amat(n),iter(3),array(n),row1(n)
c
c  Set initial value of relaxation parameter to 1.0
c
      relax = 1.0
c
c  Set iteration count to zero and initialize iteration array
c
      iflag = 0
      iter(1) = 100.0                                                   40
      iter(2) = 100.0
      iter(3) = 100.0
c
c  Set up x = u1(2) to u1(7)
c
        do 300 j = 1,n
          x(j) = u1(j+1)
 300      continue
c
c  Calculate the Jacobian matrix and range vector corresponding to this x  50
c vector
c
c  For each of the chosen observation times calculate the current Kepler
```

```
c elements and convert to current Equinoctial elements
c
 320    do 360 j = 1,n
        tto = t(j)
        call kepeqn(temp,x,retro,kep)
        call brolyd(oscele,temp,ipert,ipass,idmean,orbel,pi)
        if(intgr.eq.1)then                                              60
        comp = 10
        goto 470
        endif
        if(prop.lt.0.5)then
        do 55 k = 1,n
          array(k) = oscele(k)
 55       continue
        else
        do 60 k = 1,n
          array(k) = temp(k)                                            70
 60       continue
        endif
        call eqnkep(eqnelm,retro,array,setrtr)
c
c  Convert from Equinoctial elements to position and velocity using
c subroutine cartes. This subroutine also sets up the satelm block data
c to correspond to the current time.
c
        call cartes(pos,vel,eqnelm,retro,mu)
c                                                                       80
c  Write the current satellite position to an appropriate array.
c
        do 355 k = 1,3
          sat(k,j) = pos(k)
 355      continue
c
c  Calculate the current range and store it in the appropriate array.
c
        y(j) = sqrt((sat(1,j)-stn(1,j))**2+(sat(2,j)-stn(2,j))**
    +2+(sat(3,j)-stn(3,j))**2)                                          90
c
c  Calculate the derivative matrix either analytically in the case of 2-body
c mechanics or by numerical differencing in the case of Brouwer propagation
c
        if(prop.lt.0.5)then
c
c  Calculate the matrix of partial derivatives of ranges with respect to epoch
c mean Brouwer elements using subroutine derive
c
        call derive(row1,y(j),x,retro,kep,ipass,ipert,idmean,           100
    +pi,setrtr,mu,j,stn)
        if(intgr.eq.1)then
        comp = 10
        goto 470
        endif
        do 359 k = 1,n
          drdeq(j,k) = row1(k)
```

```
359         continue
          goto 360
          else                                                    110
c
c  Calculate the matrix of partial derivatives of current position and
c velocity with respect to current equinoctial elements using subroutine
c crtdrv.
c
          call crtdrv(dpvdeq)
c
c  Multiply the matrix of partial derivatives(dpvdeq) by the matrix of
c partial derivatives of current state in Equinoctial elements with respect to
c epoch state in Equinoctial elements, and return the resulting matrix back   120
c to dpvdeq.
c   Note that since the matrix of partial derivatives of current state in
c Equinoctial elements, with respect to epoch state in Equinoctial elements,
c is the six-by-six identity except that the (6,1) entry is :
c -1.5*sqrt(mu/a**5)*t.
c
          do 356 k = 1,n
            dpvdeq(k,1) = dpvdeq(k,1)-1.5*sqrt(mu/x(1)**5)*t(j)*d
     +pvdeq(k,6)
 356        continue                                               130
c
c  Calculate the row vector (drdpv) resulting from partial differentiation of
c range at the current time with respect to current position and velocity. The
c last three entries are zeroes since range is independent of current velocity
c components.
c
          do 357 k = 1,3
            drdpv(k) = (sat(k,j)-stn(k,j))/y(j)
 357        continue
c                                                                  140
c  Carry out the multiplication drdpv x dpvdeq to get the ith row of the
c Jacobian matrix (drdeq).
c
          do 358 k = 1,n
            drdeq(j,k) = drdpv(1)*dpvdeq(1,k)+drdpv(2)*dpvdeq(2,k
     +)+drdpv(3)*dpvdeq(3,k)
 358        continue
          endif
 360      continue
c                                                                  150
c  Set up the 7x7 system to be solved in order to get the change in u1, deltau
c
c  Calculate the 7x1 right hand side column vector, rhs. Also, test rhs for
c convergence
c
        value = 0.0
        do 370 j = 1,n
          rhs(j+1) = -yest(j)+y(j)-u1(1)*amat(j)
          value = max(value,abs(rhs(j+1)/max(abs(yex(j)),ab
     +s(yest(j)),abs(y(j)))))                                      160
 370      continue
```

```
      write(1,*)value,u1(1)
      if(value.le.eps2)then
      goto 460
      endif
c
c  Update iteration array
c
      iter(3) = iter(2)
      iter(2) = iter(1)                                                  170
      iter(1) = value
c
c  If convergence criterion is not met, update iteration number and if it is
c less than 10, update u and do the next iteration. If the iteration number is
c not less than 10 then return to the start of the bootstrap starter and add
c deltas to the next component of the initial u.
c
      iflag = iflag+1
      if(iflag.gt.10)then
      goto 440                                                           180
      endif
      rhs(1) = 0.0
c
c  Calculate the 7x7 left hand side matrix, lhs
c
      do 390 j = 1,n
        lhs(j+1,1) = amat(j)
        do 380 k = 1,n
          lhs(j+1,k+1) = -drdeq(j,k)
380       continue                                                       190
390     continue
      call tangent(amat,drdeq,row)
      do 400 j = 1,7
        lhs(1,j) = row(j)
        duds(j) = lhs(1,j)
400     continue
c
c  Solve the 7x7 system using subroutines ludcmp and lubksb which return deltau
c as rhs
c                                                                        200
      call ludcmp(lhs,7,7,indx,d)
      call lubksb(lhs,7,7,indx,rhs)
c
c  If the iterative scheme is diverging try letting the relaxation parameter
c equal 0.8
c
      if(iter(1).gt.iter(2).and.iter(2).gt.iter(3))then
      relax = 0.8
      endif
c                                                                        210
c  Add the calculated change
c
415   do 425 j = 1,n
        x(j) = x(j)+relax*rhs(j+1)
425     continue
```

```
      u1(1) = u1(1)+relax*rhs(1)
c
c  Update u1
c
 435    do 430 j = 1,n                                                  220
         u1(j+1) = x(j)
 430     continue
c
c  If eccentricity is greater than 1, write a failure message to file, let
c comp = 10 so that on returning to the main program, the last predicted
c point will be discarded and the step-size halved
c
      if((x(2)**2+x(3)**2).ge.1.0)then
      write(1,*)'Failure in corrector due to eccentricity > 1'
      comp = 10                                                        230
      goto 470
      endif
      goto 320
c
c  If the Newton-Raphson scheme converged in 10 or less iterations, then
c set comp = 0, otherwise, set comp = 10
c
 440   comp  = 10
      goto 470
 460   comp = 0                                                         240
 470   end
```

```
      subroutine critical(u1,u2,u3,s1,s2,s3,kep,pi,kflag,stn,yest,yex,
     +mu,retro,setrtr,t,scale,comp,ipass,ipert,idmean,orbel,oscele,amat,
     +prop,eps2)
c
c
c  This subroutine determines whether a local extremum in lambda has been
c passed. If so, it is estimated and then refined and stored
c
c  Required type statements
c                                                                       10
      implicit none (a-z)
      double precision u1,u2,u3,s1,s2,s3,sextrm,uextrm,slow,shigh,kep1,
     +kep,retro,pi,uhigh,ulow,stn,yest,yex,mu,retro,t,comp,eps2,scale,
     +duds,orbel,oscele,crit,value,calc1,calc2,sexnew,uexnew,amat,prop,
     +x,var
      integer i,j,kflag,iflag,n,ipass,ipert,idmean,k
      logical setrtr
c
c Set n = 6
c                                                                       20
      parameter(n=6)
c
c  Required dimension statements
c
      dimension u1(7),u2(7),u3(7),uextrm(7),kep(n),kep1(n),x(n),uhigh(7)
     +,ulow(7),yest(n),yex(n),t(n),scale(7),duds(7),stn(3,n),
     +crit(7,0:10),uexnew(7),amat(n)
c
c  Calculate the value of arc length sextrm for which the derivative of lambda
c with respect to s is 0 on the extrapolated curve and take a point either side  30
c of it to define the range in which the critical point lies
c
      calc1 = u1(1)*(s2+s3)/((s1-s2)*(s1-s3))+u2(1)*(s1+s3)/((s2-s1)*(s2
     +-s3))+u3(1)*(s1+s2)/((s3-s1)*(s3-s2))
      calc2 = u1(1)/((s1-s2)*(s1-s3))+u2(1)/((s2-s1)*(s2-s3))+u3(1)/((s3
     +-s2)*(s3-s1))
      sextrm = calc1/(2*calc2)
      if((sextrm-s1)*(sextrm-s3).gt.0)then
      write(11,*)' '
      write(11,*)'No real critical point exists here, we probably have a    40
       + turn around of the      solution curve'
      write(11,*)' '
      kflag = kflag+1
      goto 300
      endif
      shigh = sextrm+0.2*(s1-s3)
      slow = sextrm-0.2*(s1-s3)
c
c  Ensure that these three points all lie within the initial range
c                                                                       50
 30   if((shigh.gt.s3.and.shigh.gt.s1).or.(shigh.lt.s3.and.shigh.lt.s1))
     +then
      shigh = (shigh+sextrm)/2.0
```

```
      goto 30
      endif
 40   if((slow.gt.s3.and.slow.gt.s1).or.(slow.lt.s3.and.slow.lt.s1))then
      slow = (slow+sextrm)/2.0
      goto 40
      endif
c                                                                       60
c  Predict and correct these points. If the corrector fails on any of these
c points write that message to the critical point storage file
c
      call predict(sextrm,s1,s2,s3,uextrm,u1,u2,u3)
      call predict(shigh,s1,s2,s3,uhigh,u1,u2,u3)
      call predict(slow,s1,s2,s3,ulow,u1,u2,u3)
      call correct(ulow,stn,yest,yex,mu,retro,setrtr,t,iflag,comp,eps2,
     +duds,kep,ipass,ipert,idmean,pi,orbel,oscele,amat,prop)
      if(comp.gt.5)then
      write(11,*)' '                                                    70
      write(11,*)'A critical point has been passed but due to a non-conv
       +ergence condition in correct, it was prevented from being recorded
       + (ulow)'
      write(11,*)' '
      kflag = kflag+1
      goto 300
      endif
      call correct(uhigh,stn,yest,yex,mu,retro,setrtr,t,iflag,comp,eps2,
     +duds,kep,ipass,ipert,idmean,pi,orbel,oscele,amat,prop)
      if(comp.gt.5)then                                                 80
      write(11,*)' '
      write(11,*)'A critical point has been passed but due to a non-conv
       +ergence condition in correct, it was prevented from being recorded
       + (uhigh)'
      write(11,*)' '
      kflag = kflag+1
      goto 300
      endif
      call correct(uextrm,stn,yest,yex,mu,retro,setrtr,t,iflag,comp,eps2
     +,duds,kep,ipass,ipert,idmean,pi,orbel,oscele,amat,prop)          90
      if(comp.gt.5)then
      write(11,*)' '
      write(11,*)'A critical point has been passed but due to a non-conv
       +ergence condition in correct, it was prevented from being recorded
       + (uextrm)'
      write(11,*)' '
      kflag = kflag+1
      goto 300
      endif
c                                                                       100
c  Correct the arclengths of these three points
c
      call arc(shigh,s2,s3,uhigh,u2,u3)
      call arc(slow,s2,s3,ulow,u2,u3)
      call arc(sextrm,s2,s3,uextrm,u2,u3)
c
c  Check to make sure that after correcting the three points the critical
```

```
c point still lies within the range defined by the corrected points
c
      if(uextrm(1).gt.uhigh(1).and.ulow(1).gt.uextrm(1))then                    110
      write(11,*)' '
      write(11,*)'After correcting 3 points in the initial interval the
       +critical point is no longer in the reduced interval'
      write(11,*)' '
      goto 300
      endif
      if(uhigh(1).gt.uextrm(1).and.uextrm(1).gt.ulow(1))then
      write(11,*)' '
      write(11,*)'After correcting 3 points in the initial interval the
       +Critical point is no longer in the reduced interval'                    120
      write(11,*)' '
      goto 300
      endif
      if((sextrm-shigh)*(sextrm-slow).ge.0)then
      write(11,*)' '
      write(11,*)'No real critical point exists here.'
      write(11,*)' '
      goto 300
      endif
c                                                                               130
c  If the range is small enough consider uextrm to be the critical point.
c Otherwise, narrow the range further
c
 45   write(11,*)' '
      write(11,110)shigh,sextrm,slow,uhigh(1),uextrm(1),ulow(1)
      write(11,*)' '
 110  format(6(F20.10))
      if(abs(uhigh(1)-ulow(1)).gt.1.0d-7.or.abs(shigh-slow).gt.1.0d-1)
     +then
c                                                                               140
c  Calculate the value of arc length sexnew which lies half-way between slow
c and shigh. Extrapolate the curve to get a point at that arc length. Correct
c that point. Correct the arc length of the corrected point. Make sure that
c the new corrected point lies in the range containing the critical point. If
c it does, then decide which of the original three points to throw away
c
c
      sexnew = slow+0.5*(shigh-slow)
 46   call predict(sexnew,shigh,sextrm,slow,uexnew,uhigh,uextrm,ulow)
      call correct(uexnew,stn,yest,yex,mu,retro,setrtr,t,iflag,comp,eps2        150
     +,duds,kep,ipass,ipert,idmean,pi,orbel,oscele,amat,prop)
      if(comp.gt.5)then
      write(11,*)' '
      write(11,*)'A critical point has been passed but due to a non-conv
       +ergence condition in correct it was prevented from being recorded
       +(uexnew)'
      write(11,*)' '
      kflag = kflag+1
      goto 300
      endif                                                                     160
      call arc(sexnew,shigh,slow,uexnew,uhigh,ulow)
```

```
      if(sexnew.eq.sextrm)then
      sexnew = slow+0.6*(shigh-slow)
      goto 46
      endif
      write(11,*)' '
      write(11,110)sexnew,uexnew(1)
      write(11,*)' '
      if((sexnew-slow)*(sexnew-shigh).gt.0)then
      write(11,*)' '                                                    170
      write(11,*)'The value of the latest s estimate falls outside the c
       +ritical point interval'
      write(11,*)' '
      kflag = kflag+1
      goto 300
      endif
      if((uextrm(1).gt.ulow(1).and.uexnew(1).ge.uextrm(1)).or.(uextrm(1)
     +.lt.ulow(1).and.uexnew(1).le.uextrm(1)))then
      if((sexnew-sextrm)*(sexnew-slow).gt.0)then
      do 50 i = 1,7                                                     180
        ulow(i) = uextrm(i)
50      continue
      slow = sextrm
      else
      do 60 i = 1,7
        uhigh(i) = uextrm(i)
60      continue
      shigh = sextrm
      endif
      do 70 i = 1,7                                                     190
        uextrm(i) = uexnew(i)
70      continue
      sextrm = sexnew
      goto 45
      endif
      if((uextrm(1).gt.ulow(1).and.uexnew(1).lt.uextrm(1)).or.(uextrm(1)
     +.lt.ulow(1).and.uexnew(1).gt.uextrm(1)))then
      if((sextrm-slow)*(sextrm-sexnew).gt.0)then
      do 80 i = 1,7
        ulow(i) = uexnew(i)                                             200
80      continue
      slow = sexnew
      else
      do 90 i = 1,7
        uhigh(i) = uexnew(i)
90      continue
      shigh = sexnew
      endif
      goto 45
      endif                                                             210
      endif
c
c  Check to see if this critical point is the same as one which has been
c recorded previously
c
```

```
      do 88 k = 0,kflag
        value = 0.0
        do 85 j = 1,7
          if(j.le.6)then
          value = value+((crit(j,k)-uextrm(j))/scale(j))**2                    220
          goto 85
          endif
          var = abs(mod(crit(j,k),2*pi)-mod(uextrm(j),2*pi))
          if(var.gt.pi)then
          var = var-2*pi
          endif
          value = value+(var/scale(j))**2
 85        continue
        if(value.lt.1.0)then
        write(11,*)' '                                                         230
        write(11,*)'A critical point has been encountered but it is the
       + same as one which has been recorded before'
        write(11,*)' '
        goto 300
        endif
 88     continue
c
c  Convert to Kepler elements for output
c
      do 100 j = 1,6                                                           240
        x(j) = uextrm(j+1)
 100    continue
      call kepeqn(kep1,x,retro,kep)
c
c  Store new critical point in crit and update number of critical points
c
      do 95 j = 1,7
        crit(j,kflag) = uextrm(j)
 95     continue
      kflag = kflag+1                                                          250
c
c  Write critical point data to file
c
      write(11,*)' '
      write(11,*)'       One critical point is:'
      write(11,*)'                lambda =',uextrm(1),' '
      write(11,*)'                     a =',kep1(1),'km'
      write(11,*)'                     e =',kep1(2),' '
      write(11,*)'                     i =',kep1(3)*180/pi,'degrees'
      write(11,*)'                     w =',kep1(4)*180/pi,'degrees'           260
      write(11,*)'                     q =',kep1(5)*180/pi,'degrees'
      write(11,*)'                     m =',kep1(6)*180/pi,'degrees'
      write(11,*)' '
 300  comp = 0
      end
```

```
      subroutine arc(s1,s2,s3,u1,u2,u3)
c
c
c  This subroutine corrects the value of the arc length along the
c solution curve from start point to current point
c
c  Required type statements
c
      implicit none (a-z)
      double precision a,b,c,d,quad,s1,s2,s3,u1,u2,u3,dL1ds,dL2ds,dL3ds,
     +duds,value,s1old
      integer i,j
c
c  Dimension the various arrays used
c
      dimension quad(3),u1(7),u2(7),u3(7),duds(7)
c
c  Set up various required constants
c
      a = 0.555555555555556
      b = 0.225403330758517
      c = 1.774596669241483
      d = 0.888888888888889
c
c  Calculate the quadrature evaluation points
c
 20   s1old = s1
      quad(1) = 0.5*(b*s2+c*s1)
      quad(2) = 0.5*(s2+s1)
      quad(3) = 0.5*(c*s2+b*s1)
c
c  For each of the evaluation points, evaluate the function, i.e., the root
c mean square of the components of the gradient vector at the evaluation
c point
c
      do 100 i = 1,3
        dL1ds = ((quad(i)-s3)+(quad(i)-s2))/((s1-s2)*(s1-s3))
        dL2ds = ((quad(i)-s3)+(quad(i)-s1))/((s2-s1)*(s2-s3))
        dL3ds = ((quad(i)-s1)+(quad(i)-s2))/((s3-s2)*(s3-s1))
        value = 0.0
        do 50 j = 1,7
          duds(j) = dL1ds*u1(j)+dL2ds*u2(j)+dL3ds*u3(j)
          value = value+duds(j)**2
 50       continue
        quad(i) = sqrt(value)
 100    continue
c
c  Calculate the corrected curve length
c
      s1 = s2+(s1-s2)*(d*quad(2)+a*quad(1)+a*quad(3))/2.0
c
c  Test to see if the calculated change in curve length exceeds some tolerance.
c If it does then run it through the arc length corrector again
```

```
c
      if(abs(s1-s1old).gt.1.0d-8)then
      goto 20
      endif
      end
```

```
      subroutine terminate(u1,u2,u3,s1,s2,s3,stn,yest,yex,mu,retro,
     +setrtr,t,xest,comp,scale,kep,eps2,lambda,mflag,ipass,ipert,idmean,
     +pi,orbel,oscele,number,amat,prop)
c
c
c  This subroutine checks to see if the solution curve has returned to the
c initial point by completing the solution loop
c
c  Required type statements
c                                                                       10
      implicit none (a-z)
      logical setrtr
      double precision u1,u2,u3,uint,s1,s2,s3,sint,stn,uup,kep,kep1,
     +udown,yest,yex,mu,t,ua,u,xest,scale,x,comp,retro,var,
     +value,pi,eps2,duds,lambda,pi,orbel,oscele,L1,L2,amat,prop
      integer i,j,n,iflag,mflag,ipass,ipert,idmean,number
c
c  Set n = 6
c
      parameter(n=6)                                                    20
c
c  Required dimension statements
c
      dimension u1(7),u2(7),u3(7),uint(7,0:50),sint(0:50),xest(n),
     +stn(3,n),yest(n),yex(n),t(n),uup(7),udown(7),ua(7),u(7),scale(7),
     +kep(n),kep1(n),x(n),duds(7),orbel(5),oscele(n),amat(n)
      comp = 0
c
c  Let uint(1:7,0) = u2(1:7), uint(1:7,50) = u1(1:7), sint(50) = s1, and
c sint(0) = s2                                                          30
c
      do 25 i = 1,7
        uint(i,0) = u2(i)
        uint(i,50) = u1(i)
 25     continue
      sint(0) = s2
      sint(50) = s1
c
c  Otherwise, calculate 49 points on the solution curve evenly spaced between
c s1 and s2 using linear interpolation if the algorithm has only two points   40
c available, or using quadratic interpolation otherwise.
c
      if(number.eq.1)then
      do 40 i = 1,49
        sint(i) = s2+i*(s1-s2)/50.0
        L1 = (sint(i)-s2)/(s1-s2)
        L2 = (sint(i)-s1)/(s2-s1)
        do 30 j = 1,7
          uint(j,i) = L1*u1(j)+L2*u2(j)
 30       continue                                                      50
 40     continue
      goto 80
      endif
```

```
      do 75 i = 1,49
        sint(i) = s2+i*(s1-s2)/50.0
        call predict(sint(i),s1,s2,s3,u,u1,u2,u3)
        do 50 j = 1,7
          uint(j,i) = u(j)
 50       continue
 75     continue                                                    60
c
c
c  Pick out pairs of these predicted solutions that straddle the lambda = 0
c hyperplane.
c
c
 80   do 200 i = 1,50
        if((uint(1,i))*(uint(1,i-1)).ge.0)then
        goto 200
        endif                                                      70
c
c  For each such pair, correct them using N-R. If corrector does not converge
c set comp = 10 and return to the main program
c
        do 125 j = 1,7
          uup(j) = uint(j,i)
          udown(j) = uint(j,i-1)
 125      continue
        if(i.eq.1)then
        goto 130                                                   80
        endif
        call correct(udown,stn,yest,yex,mu,retro,setrtr,t,iflag,comp,
     +eps2,duds,kep,ipass,ipert,idmean,pi,orbel,oscele,amat,prop)
        if(comp.gt.5)then
        goto 300
        endif
        if(i.eq.50)then
        goto 140
        endif
 130    call correct(uup,stn,yest,yex,mu,retro,setrtr,t,iflag,comp,eps2  90
     +,duds,kep,ipass,ipert,idmean,pi,orbel,oscele,amat,prop)
        if(comp.gt.5)then
        goto 300
        endif
c
c  Let ua be a linearly interpolated state at lambda = 0
c
 140    do 150 j = 1,7
          ua(j) = udown(j)-udown(1)*(uup(j)-udown(j))/(uup(1)-ud
     +own(1))                                                      100
 150      continue
c
c  Correct ua using N-R. If it does not converge set comp = 10 and return to
c the main program
c
        call correct(ua,stn,yest,yex,mu,retro,setrtr,t,iflag,comp,eps2,
     +duds,kep,ipass,ipert,idmean,pi,orbel,oscele,amat,prop)
```

```
      if(comp.gt.5)then
      goto 300
      endif                                                          110
c
c  If the corresponding lambda is not equal to 0 within the specified
c tolerance let either uup or udown, whichever is further from 1, equal ua,
c and repeat
c
      if(abs(ua(1)).gt.0.00001)then
      if(abs(uup(1)).gt.abs(udown(1)))then
      do 175 j = 1,7
        uup(j) = ua(j)
175     continue                                                     120
      goto 140
      endif
      do 190 j = 1,7
        udown(j) = ua(j)
190     continue
      goto 140
      endif
c
c
c  If lambda is equal to 0 within the specified tolerance check for return to   130
c the initial state. If it has returned, and if either the curve approached
c from the opposite direction from which it left or if this is the second
c time the curve returned to the initial point, terminate algorithm.
c Otherwise, do next step.
c
c
      value = 0.0
      do 192 j = 1,6
        if(j.le.5)then
        value = value+((xest(j)-ua(j+1))/scale(j+1))**2              140
        goto 192
        endif
        var = abs(mod(xest(j),2*pi)-mod(ua(j+1),2*pi))
        if(var.gt.pi)then
        var = var-2*pi
        endif
        value = value+(var/scale(j+1))**2
192     continue
      write(1,*)' '
      write(1,*)value                                                150
      write(1,*)(xest(j), j = 1,6)
      write(1,*)(ua(j), j = 2,7)
      write(1,*)' '
      if(value.lt.1.0d-2)then
      mflag = mflag+1
      if(mflag.gt.1.or.lambda*u1(1).gt.0)then
c
c  Convert to Kepler elements for output
c
      do 195 j = 1,n                                                 160
        x(j) = ua(j+1)
```

```
195       continue
      call kepeqn(kep1,x,retro,kep)
c
c  Write termination point to file
c
      write(1,*)'Termination occurred at:'
      write(1,*)'                    lambda =',ua(1),' '
      write(1,*)'                         a =',kep1(1),'km'
      write(1,*)'                         e =',kep1(2),' '                    170
      write(1,*)'                         i =',kep1(3)*180/pi,'degrees'
      write(1,*)'                         w =',kep1(4)*180/pi,'degrees'
      write(1,*)'                         q =',kep1(5)*180/pi,'degrees'
      write(1,*)'                         m =',kep1(6)*180/pi,'degrees'
      comp = 20
      goto 300
      endif
      endif
200    continue
300  end                                                                    180
```

```
      subroutine stpsz(iflag,deltas)
c
c  This subroutine chooses the next step-size based on the last step size and
c on how many iterations it took to converge
c
c
c  Required type statements
c
      implicit none (a-z)
      double precision deltas                                          10
      integer iflag
c
c  If the last step took 9 or 10 steps, leave deltas to 100% of the value
c it had for the last step. If the last step took 8 or less steps, increase
c deltas to 140% of the value it had for the last step
c
      if(iflag.le.8)then
      deltas = 1.4*deltas
      endif
      end                                                              20
```

```
      subroutine derive(row1,range,x,retro,kep,ipass,ipert,idmean,
     +pi,setrtr,mu,j,stn)
c
c
c  This subroutine numerically calculates the derivatives of ranges with
c respect to epoch mean Brouwer elements
c
c
c  Required type statements
c                                                                       10
      implicit none (a-z)
      double precision row1,range,x,delta,scale,tto,temp,kep,oscele,
     +orbel,pi,retro,eqnelm,pos,vel,mu,y,stn,r8dum
      logical setrtr
      integer j,i,ipass,ipert,idmean,intgr
c
c  Required common block
c
      common/dcint /r8dum(126),tto,intgr
c                                                                       20
c Dimension local arrays
c
      dimension row1(6),x(6),scale(6),temp(6),kep(6),oscele(6),orbel(5),
     +eqnelm(6),pos(3),vel(3),y(6),stn(3,6)
c
c  Set up scaling of Equinoctial elements
c
      scale(1) = 1000
      scale(2) = 0.001
      scale(3) = 0.001                                                  30
      scale(4) = 1
      scale(5) = 1
      scale(6) = 1
c
c  Propagate the element set with scaled delta added to each component in
c turn in order to get range differences
c
      delta = 1.0d-8
      do 100 i = 1,6
        x(i) = x(i)+delta*scale(i)                                      40
        call kepeqn(temp,x,retro,kep)
        call brolyd(oscele,temp,ipert,ipass,idmean,orbel,pi)
        if(intgr.eq.1)then
        goto 200
        endif
        call eqnkep(eqnelm,retro,oscele,setrtr)
        call cartes(pos,vel,eqnelm,retro,mu)
        y(i) = sqrt((pos(1)-stn(1,j))**2+(pos(2)-stn(2,j))**2+(pos(3)-s
     +tn(3,j))**2)
        row1(i) = (y(i)-range)/(delta*scale(i))                         50
        x(i) = x(i)-delta*scale(i)
  100   continue
  200   end
```

```
      subroutine tangent(amat,drdeq,row)
c
c
c  This subroutine calculates the tangent vector of the solution curve
c
c  Required type statements
c
      implicit none (a-z)
      double precision amat,drdeq,row,inv,dxds,trans,column,G
      integer j,k,n                                                    10
c
c  Fix the value of n to be 6
c
      parameter(n=6)
c
c  Required dimension statements
c
      dimension amat(n),drdeq(n,n),row(7),inv(n,n),dxds(n),trans(n,n),
     +column(n)
c                                                                      20
c  Invert Jacobian and let inv be the negative of it
c
      call invert(drdeq,inv)
      do 392 j = 1,n
        do 391 k = 1,n
          inv(j,k) = -inv(j,k)
 391      continue
 392    continue
c
c  Do matrix multiplication of inv by amat and store in dxds          30
c
      do 394 j = 1,n
        dxds(j) = 0.0
        do 395 k = 1,n
          dxds(j) = dxds(j)+inv(j,k)*amat(k)
 395      continue
 394    continue
c
c  Take transpose of inv and store in trans
c                                                                      40
      do 397 j = 1,n
        do 396 k = 1,n
          trans(j,k) = inv(k,j)
 396      continue
 397    continue
c
c  Do matrix multiplication of trans by dxds and store in column
c
      do 399 j = 1,n
        column(j) = 0.0                                                50
        do 398 k = 1,n
          column(j) = column(j)+trans(j,k)*dxds(k)
 398      continue
```

```
 399    continue
c
c  Calculate G
c
      G = 0.0
      do 401 j = 1,n
        G = G+amat(j)*column(j)                                        60
 401    continue
      G = sqrt(G+1.0)
c
c  Calculate the tangent vector and return it in row
c
      row(1) = 1.0/G
      do 402 j = 1,n
        row(j+1) = -dxds(j)/G
 402    continue
      end                                                              70
```

```
      subroutine invert(a,y)
c
c
c  This subroutine calculates the inverse of the matrix a which it receives
c in the call argument, and returns that inverse as y.
c
c  Necessary type statements
c
      implicit none (a-z)
      double precision a,y,indx,d                                          10
      integer i,j,n
c
c  Fix the value of n to be 6
c
      parameter(n=6)
c
c  Necessary dimension statements
c
      dimension a(n,n),y(n,n),indx(n)
c                                                                          20
c
c  Initializes y to be the identity matrix.
c
c
      do 12 i = 1,n
        do 11 j = 1,n
          y(i,j) = 0.0
 11       continue
        y(i,i) = 1.0
 12     continue                                                          30
c
c
c  Does the matrix inversion using the LINPAK subroutines ludcmp and lubksb.
c
c
      call ludcmp(a,n,n,indx,d)
      do 13 j = 1,n
        call lubksb(a,n,n,indx,y(1,j))
 13     continue
      end                                                                40
```

```
      BLOCK DATA
C
C
C
C     /ELIPSD/ ******************************************************
C
C     Size and shape parameters for central-body ellipsoid.
C
C        SEMMAJ  O    Equatorial radius.
C        SEMMIN  O    Polar radius.                                      10
C        ECCEN   O    Eccentricity.
C        ESQR    O    ECCEN ** 2
C        RTESQR  O    SQRT ( 1 - ECCEN ** 2)
C        FLAT    O    Flattening coefficient.
C        FLTINV  O    Inverse flattening coefficient.
C
C     Switches.
C
C        INIELL  O    Initialize central-body ellipsoid?
C        PRTELL  O    Iteration print?                                   20
C
C
C
C**************** HISTORY ********************************************
C
C
C
C     VERSION:  October 1988
C        Fortran block data subprogram for the IBM 3090.
C                                                                        30
C     ANALYSIS
C        Leo W. Early, Jr.          -- Charles Stark Draper Laboratory
C
C     PROGRAMMER
C        Leo W. Early, Jr.          -- Charles Stark Draper Laboratory
C
C
C
C**************** DECLARATIONS ***************************************
C                                                                        40
C
C
      IMPLICIT        DOUBLE PRECISION   (A-H,O-Z)
C
C     Data Types ==================================================
C
      LOGICAL         INIELL     ,PRTELL     ,LELIPS
C
C     /ELIPSD/ ====================================================
C                                                                        50
      COMMON /ELIPSD/  RELIPS (7)           ,LELIPS (2)
      EQUIVALENCE     (SEMMAJ               ,RELIPS (1)          )
      EQUIVALENCE     (SEMMIN               ,RELIPS (2)          )
```

```
      EQUIVALENCE    (ECCEN          ,RELIPS (3)       )
      EQUIVALENCE    (ESQR           ,RELIPS (4)       )
      EQUIVALENCE    (RTESQR         ,RELIPS (5)       )
      EQUIVALENCE    (FLAT           ,RELIPS (6)       )
      EQUIVALENCE    (FLTINV         ,RELIPS (7)       )
      EQUIVALENCE    (INIELL         ,LELIPS (1)       )
      EQUIVALENCE    (PRTELL         ,LELIPS (2)       )                    60
C
C
C
C***************** DATA STATEMENTS ************************************
C
C
C
C           Equatorial radius.  (km)
C
      DATA  SEMMAJ  /  6378.135  D 0  /                                      70
C
C           Inverse flattening coefficient.
C
      DATA  FLTINV  /  298.26    D 0  /
C
C
C
C           Initialize central-body ellipsoid?
C
      DATA  INIELL  /  .TRUE.    /                                           80
C
C           Iteration print?
C
      DATA  PRTELL  /  .FALSE.   /
      END
```

```
      BLOCK DATA
C
C
C
C           ********
C           /SATELM/
C           ********
C
C
C                                                                       10
C     1.  SATELLITE POSITION AND VELOCITY.
C
C     2.  SATELLITE EQUINOCTIAL ORBIT ELEMENTS.
C
C     3.  PARTIAL DERIVATIVES OF EQUINOCTIAL ORBIT ELEMENTS WITH
C         RESPECT TO VELOCITY.
C
C     4.  AUXILIARY PARAMETERS.
C
C                                                                       20
C
C           ***********
C           DESCRIPTION
C           ***********
C
C
C
C     THIS COMMON BLOCK CONTAINS THE ORBIT ELEMENTS CURRENTLY BEING
C     USED.  THE CONTENTS WILL CHANGE WHENEVER ANY OF THE FOLLOWING
C     SUBROUTINES IS CALLED:                                            30
C
C        CARTES                  AUXEQN
C        QDRCRT                  AUXDRV
C
C
C
C***************** TABLE OF CONTENTS *********************************
C
C
C                                                                       40
C  VARIABLE  DIMENSION  LOCATION  DESCRIPTION
C
C       POSITION AND VELOCITY.
C
C  X        3       1     POSITION.
C  V        3       4     VELOCITY.
C
C       AUXILIARY PARAMETERS FOR POSITION AND VELOCITY.
C
C  XORB     2       7     POSITION IN ORBITAL REFERENCE FRAME.           50
C  VORB     2       9     VELOCITY IN ORBITAL REFERENCE FRAME.
C  RDA      1      11     RADIAL DISTANCE / SEMIMAJOR AXIS
C
```

```
C     SATELLITE LONGITUDE.  (TWO OPTIONS)
C
C  F       1     12    ECCENTRIC LONGITUDE.
C  SINF    1     13    SIN (F)
C  COSF    1     14    COS (F)
C  DLAMDF  1     87    D LAMBDA / D F
C                                                                      60
C  XL      1     12    TRUE LONGITUDE:  L
C  SINL    1     13    SIN (L)
C  COSL    1     14    COS (L)
C  DLAMDL  1     87    D LAMBDA / D L
C
C     EQUINOCTIAL ORBIT ELEMENTS.
C
C  A       1     15    SEMIMAJOR AXIS.
C  XH      1     16    H
C  XK      1     17    K                                               70
C  P       1     18    P
C  Q       1     19    Q
C  XLAMDA  1     20    MEAN LONGITUDE:  LAMBDA
C
C  RETRG   1     21    RETROGRADE FACTOR.
C                       1  DIRECT ELEMENTS
C                      -1  RETROGRADE ELEMENTS
C
C     AUXILIARY PARAMETERS FOR EQUINOCTIAL ORBIT ELEMENTS.
C                                                                      80
C  FAXIS   3     22    F AXIS OF ORBITAL REFERENCE FRAME.
C  GAXIS   3     25    G AXIS OF ORBITAL REFERENCE FRAME.
C  WAXIS   3     28    W AXIS OF ORBITAL REFERENCE FRAME.
C
C  XMU     1     31    GRAVITATIONAL CONSTANT OF CENTRAL BODY.
C  XMEAN   1     32    KEPLER MEAN MOTION:  N
C  XNA     1     33    N*A
C  XNA2    1     34    N*A*A
C
C  XHH     1     35    H*H                                             90
C  XKK     1     36    K*K
C  HHKK    1     37    H*H + K*K
C  ONEHK   1     38    1 - H*H - K*K
C  SRTHK   1     39    SQRT (1 - H*H - K*K)
C  BETAUX  1     40    1 / (1 + SQRT (1 - H*H - K*K))
C  ONEHHB  1     41    1 - H*H*BETAUX
C  ONEKKB  1     42    1 - K*K*BETAUX
C  HKBETA  1     43    H*K*BETAUX
C
C  PP      1     44    P*P                                             100
C  QQ      1     45    Q*Q
C  PPQQ    1     46    P*P + Q*Q
C  ONEPQ   1     47    1 + P*P + Q*Q
C  ONEPQI  1     48    1 / (1 + P*P + Q*Q)
C
C     PARTIAL DERIVATIVES.
C
```

```
C  DELMDV   6,3    49    PARTIAL DERIVATIVES OF EQUINOCTIAL ORBIT
C                      ELEMENTS WITH RESPECT TO VELOCITY.
C                                                                      110
C  (1,J)               D A       /  D V(J)
C  (2,J)               D H       /  D V(J)
C  (3,J)               D K       /  D V(J)
C  (4,J)               D P       /  D V(J)
C  (5,J)               D Q       /  D V(J)
C  (6,J)               D LAMBDA  /  D V(J)
C
C     AUXILIARY PARAMETERS FOR PARTIAL DERIVATIVES.
C
C  XMUI     1     67    1 / XMU                                         120
C  TWON2A   1     68    2 / N*N*A
C  TWONA    1     69    2 / N*A
C  TWONA2   1     70    2 / N*A*A
C
C  SRTHKI   1     71    1 / SQRT (1 - H*H - K*K)
C  ONEHKI   1     72    1 / (1 - H*H - K*K)
C  BETAH    1     73    BETAUX*H
C  BETAK    1     74    BETAUX*K
C
C  XNA2RT   1     75    XNA2*SRTHK                                      130
C  XHNA2R   1     76    H / XNA2*SRTHK
C  XKNA2R   1     77    K / XNA2*SRTHK
C
C  RTDNA    1     78    SRTHK / N*A
C  RTDNA2   1     79    SRTHK / N*A*A
C  SRTAUX   1     80    SRTHK / XNA2*(1 + SRTHK)
C  XHSRTA   1     81    H*SRTHK / XNA2*(1 + SRTHK)
C  XKSRTA   1     82    K*SRTHK / XNA2*(1 + SRTHK)
C
C  PQAUX    1     83    (1 + P*P + Q*Q) / 2*XNA2*SRTHK                  140
C
C     ORBIT DESCRIPTION PARAMETERS.
C
C  ECCEN    1     84    ECCENTRICITY.
C  PERRAD   1     85    PERIGEE RADIUS.
C  APORAD   1     86    APOGEE RADIUS.
C
C
C
C***************** HISTORY ********************************************* 150
C
C
C
C   VERSION OF FEBRUARY 1984
C      FORTRAN SUBROUTINE FOR THE IBM 3081 AND 3033.
C
C   ANALYSIS
C      LEO W. EARLY, JR.       -- CHARLES STARK DRAPER LABORATORY
C
C   PROGRAMMER                                                         160
C      LEO W. EARLY, JR.       -- CHARLES STARK DRAPER LABORATORY
```

```
C
C
C
C***************** DECLARATIONS ***************************************
C
C
C
      IMPLICIT       DOUBLE PRECISION (A-H,O-Z)
C                                                                       170
C     DIMENSIONS ********************************************************
C
      DIMENSION      X (3)              ,XORB (2)
      DIMENSION      V (3)              ,VORB (2)
      DIMENSION      FAXIS (3)            ,WAXIS (3)
      DIMENSION      GAXIS (3)            ,DELMDV (6,3)
C
C     /SATELM/ **********************************************************
C
      COMMON /SATELM/  RSATEL (87)                                      180
C
C             POSITION AND VELOCITY.
C
      EQUIVALENCE    (X     (1)          ,RSATEL (1)          ),
     *              (V     (1)          ,RSATEL (4)          )
C
C             AUXILIARY PARAMETERS FOR POSITION AND VELOCITY.
C
      EQUIVALENCE    (XORB  (1)          ,RSATEL (7)          ),
     *              (VORB  (1)          ,RSATEL (9)          ),      190
     *              (RDA                ,RSATEL (11)         )
C
C             SATELLITE LONGITUDE.  (TWO OPTIONS)
C
      EQUIVALENCE    (F                  ,RSATEL (12)         ),
     *              (SINF               ,RSATEL (13)         ),
     *              (COSF               ,RSATEL (14)         ),
     *              (DLAMDF               ,RSATEL (87)         )
      EQUIVALENCE    (XL                 ,RSATEL (12)         ),
     *              (SINL               ,RSATEL (13)         ),      200
     *              (COSL               ,RSATEL (14)         ),
     *              (DLAMDL               ,RSATEL (87)         )
C
C             EQUINOCTIAL ORBIT ELEMENTS.
C
      EQUIVALENCE    (A                  ,RSATEL (15)         ),
     *              (XH                 ,RSATEL (16)         ),
     *              (XK                 ,RSATEL (17)         ),
     *              (P                  ,RSATEL (18)         ),
     *              (Q                  ,RSATEL (19)         ),      210
     *              (XLAMDA               ,RSATEL (20)         ),
     *              (RETRG                ,RSATEL (21)         )
C
C             AUXILIARY PARAMETERS FOR EQUINOCTIAL ORBIT ELEMENTS.
C
```

```
      EQUIVALENCE    (FAXIS (1)        ,RSATEL (22)        ),
     *               (GAXIS (1)        ,RSATEL (25)        ),
     *               (WAXIS (1)        ,RSATEL (28)        ),
     *               (XMU              ,RSATEL (31)        ),
     *               (XMEAN            ,RSATEL (32)        ),                  220
     *               (XNA              ,RSATEL (33)        ),
     *               (XNA2             ,RSATEL (34)        ),
     *               (XHH              ,RSATEL (35)        ),
     *               (XKK              ,RSATEL (36)        ),
     *               (HHKK             ,RSATEL (37)        )
      EQUIVALENCE    (ONEHK            ,RSATEL (38)        ),
     *               (SRTHK            ,RSATEL (39)        ),
     *               (BETAUX           ,RSATEL (40)        ),
     *               (ONEHHB           ,RSATEL (41)        ),
     *               (ONEKKB           ,RSATEL (42)        ),                  230
     *               (HKBETA           ,RSATEL (43)        ),
     *               (PP               ,RSATEL (44)        ),
     *               (QQ               ,RSATEL (45)        ),
     *               (PPQQ             ,RSATEL (46)        ),
     *               (ONEPQ            ,RSATEL (47)        ),
     *               (ONEPQI           ,RSATEL (48)        )
C
C              PARTIAL DERIVATIVES.
C
      EQUIVALENCE    (DELMDV (1,1)     ,RSATEL (49)        )                   240
C
C              AUXILIARY PARAMETERS FOR PARTIAL DERIVATIVES.
C
      EQUIVALENCE    (XMUI             ,RSATEL (67)        ),
     *               (TWON2A           ,RSATEL (68)        ),
     *               (TWONA            ,RSATEL (69)        ),
     *               (TWONA2           ,RSATEL (70)        ),
     *               (SRTHKI           ,RSATEL (71)        ),
     *               (ONEHKI           ,RSATEL (72)        ),
     *               (BETAH            ,RSATEL (73)        ),                  250
     *               (BETAK            ,RSATEL (74)        )
      EQUIVALENCE    (XNA2RT           ,RSATEL (75)        ),
     *               (XHNA2R           ,RSATEL (76)        ),
     *               (XKNA2R           ,RSATEL (77)        ),
     *               (RTDNA            ,RSATEL (78)        ),
     *               (RTDNA2           ,RSATEL (79)        ),
     *               (SRTAUX           ,RSATEL (80)        ),
     *               (XHSRTA           ,RSATEL (81)        ),
     *               (XKSRTA           ,RSATEL (82)        ),
     *               (PQAUX            ,RSATEL (83)        )                   260
C
C     ORBIT DESCRIPTION PARAMETERS.
C
      EQUIVALENCE    (ECCEN            ,RSATEL (84)        ),
     *               (PERRAD           ,RSATEL (85)        ),
     *               (APORAD           ,RSATEL (86)        )
C
C
C
```

```
C***************** DATA STATEMENTS ************************************      270
C
C
C
C           THIS DATA STATEMENT IS INCLUDED TO PREVENT COMPIL-
C           ATION ERRORS.  IT DOES NOT AFFECT EXECUTION.
C
      DATA RDA       /  0.D0  /
      END
```

```
      SUBROUTINE  ELLGEO  (RADIUS,FLATIN,PRINT)
C
C
C
C           ********
C           FUNCTION
C           ********
C
C
C                                                                       10
C     This subroutine sets size and shape parameters for the reference
C     ellipsoid of the geodetic coordinate system.
C
C
C
C     CALLING SEQUENCE ************************************************
C
C        CALL  ELLGEO  (RADIUS,FLATIN,PRINT)
C
C                                                                       20
C
C     PARAMETERS ******************************************************
C
C        RADIUS I   Equatorial radius.
C        FLATIN I   Inverse flattening coefficient.
C        PRINT  I   Print iterated values of reduced latitude when
C                   converting from Cartesian to geodetic coordi-
C                   nates?
C
C                                                                       30
C
C     /ELIPSD/ ********************************************************
C
C     Size and shape parameters for central-body ellipsoid.
C
C        SEMMAJ O   Equatorial radius.
C        SEMMIN O   Polar radius.
C        ECCEN  O   Eccentricity.
C        ESQR   O   ECCEN ** 2
C        RTESQR O   SQRT ( 1 - ECCEN ** 2)                              40
C        FLAT   O   Flattening coefficient.
C        FLTINV O   Inverse flattening coefficient.
C
C     Switches.
C
C        INIELL O   Initialize central-body ellipsoid?
C        PRTELL O   Iteration print?
C
C
C                                                                       50
C     NO SUBROUTINES CALLED *******************************************
C
C
```

```
C
C***************** HISTORY ********************************************
C
C
C
C    VERSION:  October 1988
C        Fortran subroutine for the IBM 3090.                                  60
C
C    ANALYSIS
C        Leo W. Early, Jr.          -- Charles Stark Draper Laboratory
C
C    PROGRAMMER
C        Leo W. Early, Jr.          -- Charles Stark Draper Laboratory
C
C
C
C***************** DECLARATIONS ***************************************      70
C
C
C
      IMPLICIT        DOUBLE PRECISION (A-H,O-Z)
C
C     Data Types ===================================================
C
      LOGICAL         PRINT      ,INIELL    ,PRTELL    ,LELIPS
C
C     /ELIPSD/ ======================================================== 80
C
      COMMON /ELIPSD/  RELIPS (7)          ,LELIPS (2)
      EQUIVALENCE     (SEMMAJ              ,RELIPS (1)          )
      EQUIVALENCE     (SEMMIN              ,RELIPS (2)          )
      EQUIVALENCE     (ECCEN               ,RELIPS (3)          )
      EQUIVALENCE     (ESQR                ,RELIPS (4)          )
      EQUIVALENCE     (RTESQR              ,RELIPS (5)          )
      EQUIVALENCE     (FLAT                ,RELIPS (6)          )
      EQUIVALENCE     (FLTINV              ,RELIPS (7)          )
      EQUIVALENCE     (INIELL              ,LELIPS (1)          )      90
      EQUIVALENCE     (PRTELL              ,LELIPS (2)          )
C
C
C
C***************** BEGIN PROGRAM **************************************
C
C
C
C             **********
C             READ INPUT                                                    100
C             **********
C
C
C
C             Equatorial radius.
C
      SEMMAJ  =  RADIUS
```

```
C
C           Inverse flattening coefficient.
C                                                                    110
      FLTINV  =  FLATIN
C
C           Iteration print?
C
      PRTELL  =  PRINT
C
C
C
C           ************************
C           FIND ELLIPSOID PARAMETERS                                120
C           ************************
C
C
C
C           Eccentricity parameters.
C
      FLAT    =  1.D0 / FLTINV
      RTESQR  =  1.D0 - FLAT
      ESQR    =  (2.D0 - FLAT) * FLAT
      ECCEN   =  SQRT (ESQR)                                         130
C
C           Semiminor axis.
C
      SEMMIN  =  SEMMAJ * RTESQR
C
C
C
C           ******************
C           END INITIALIZATION
C           ******************                                       140
C
C
C
      INIELL  =  .FALSE.
      RETURN
      END
```

```
      SUBROUTINE  CRTGEO  (X,Y,Z,  HEIGHT,GLAT,GLON)
C
C
C
C           ********
C           FUNCTION
C           ********
C
C
C                                                                       10
C     This subroutine transforms geodetic latitude, longitude, and
C     height to earth-fixed Cartesian coordinates.
C
C
C
C     CALLING SEQUENCE ************************************************
C
C        CALL  CRTGEO  (X,Y,Z,  HEIGHT,GLAT,GLON)
C
C                                                                       20
C
C     PARAMETERS ******************************************************
C
C        X      O   X coordinate.
C        Y      O   Y coordinate.
C        Z      O   Z coordinate.
C
C        HEIGHT I   Height above the reference ellipsoid.
C        GLAT   I   Geodetic latitude.
C        GLON   I   Geodetic longitude.                                 30
C
C
C
C     /ELIPSD/ ********************************************************
C
C     Size and shape parameters for central-body ellipsoid.
C
C        SEMMAJ I   Equatorial radius.
C        SEMMIN I   Polar radius.
C        ECCEN  I   Eccentricity.                                       40
C        ESQR   I   ECCEN ** 2
C        RTESQR I   SQRT ( 1  -  ECCEN ** 2)
C        FLAT   I   Flattening coefficient.
C        FLTINV I   Inverse flattening coefficient.
C
C     Switches.
C
C        INIELL I   Initialize central-body ellipsoid?
C        PRTELL I   Iteration print?
C                                                                       50
C
C
C     NO SUBROUTINES CALLED *******************************************
```

```
C
C
C
C***************** HISTORY ********************************************
C
C
C                                                                       60
C    VERSION:  October 1988
C       Fortran subroutine for the IBM 3090.
C
C    ANALYSIS
C       Leo W. Early, Jr.          -- Charles Stark Draper Laboratory
C
C    PROGRAMMER
C       Leo W. Early, Jr.          -- Charles Stark Draper Laboratory
C
C                                                                       70
C
C***************** DECLARATIONS ***************************************
C
C
C
      IMPLICIT        DOUBLE PRECISION (A-H,O-Z)
C
C     Data Types ==========================================
C
      LOGICAL         INIELL     ,PRTELL     ,LELIPS                  80
C
C     /ELIPSD/ ============================================
C
      COMMON /ELIPSD/  RELIPS (7)          ,LELIPS (2)
      EQUIVALENCE     (SEMMAJ             ,RELIPS (1)          )
      EQUIVALENCE     (SEMMIN             ,RELIPS (2)          )
      EQUIVALENCE     (ECCEN              ,RELIPS (3)          )
      EQUIVALENCE     (ESQR               ,RELIPS (4)          )
      EQUIVALENCE     (RTESQR             ,RELIPS (5)          )
      EQUIVALENCE     (FLAT               ,RELIPS (6)          )   90
      EQUIVALENCE     (FLTINV             ,RELIPS (7)          )
      EQUIVALENCE     (INIELL             ,LELIPS (1)          )
      EQUIVALENCE     (PRTELL             ,LELIPS (2)          )
C
C
C
C***************** BEGIN PROGRAM **************************************
C
C
C                                                                      100
C           **********
C           INITIALIZE
C           **********
C
C
C
      IF  (INIELL)  THEN
```

```
      CALL  ELLGEO  (SEMMAJ,FLTINV,PRTELL)
    END IF
C                                                                      110
C
C
C
C           ****************
C           FIND COORDINATES
C           ****************
C
C
C
C           Sines and cosines of geodetic latitude.
C                                                                      120
    SINLAT  =  SIN (GLAT)
    COSLAT  =  COS (GLAT)
C
C           Reduced latitude.
C
    THETA   =  ATAN2 (RTESQR*SINLAT, COSLAT)
C
C           Cylindrical coordinates.
C
    R  =  SEMMAJ * COS (THETA)  +  HEIGHT * COSLAT                     130
    Z  =  SEMMIN * SIN (THETA)  +  HEIGHT * SINLAT
C
C           Cartesian coordinates.
C
    X  =  R * COS (GLON)
    Y  =  R * SIN (GLON)
    RETURN
    END
```

```
      SUBROUTINE BROLYD(OSCELE,DPELE,IPERT,IPASS,IDMEAN,ORBEL,PI)
C*********************************************************************
C*  REF. '' BROUWER-LYDDANE ORBIT GENERATOR ROUTINE''                *
C*              (X-553-70-223)                                       *
C*            BY E.A. GALBREATH 1970                                 *
C*********************************************************************
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      DOUBLE PRECISION MU
      COMMON/DCINT /R8DUM(126),TTO,intgr
      COMMON/   FRC/ DFRC(1300)                                       10
      DIMENSION OSCELE(6),DPELE(6),ORBEL(5)
      EQUIVALENCE    (BJ2,DFRC(356))     ,(BJ3,DFRC(357))     ,
     +             (BJ4,DFRC(358))     ,(BJ5,DFRC(359))     ,
     +             (DFRC(2),MU)        ,(AE,DFRC(24))
      Data BMU,RE/1.0D0,1.0D0/,BKSUBC/0.01D0/
      intgr = 0
      if(dpele(1).lt.0.or.dpele(2).ge.1.0)then
      intgr = 1
      write(1,*)'Failure in brolyd due to eccentricity > 1 or semi-major
       + axis < 0'                                                    20
      goto 1000
      endif
      EK = DSQRT(MU/AE**3)
      DELT = EK*TTO
      GO TO (10,111 ), IPASS
CI------------------------------I
CI EPOCH ELEMENTS AT EPOCH TIME I
CI------------------------------I
   10 ADP = DPELE(1)/AE
      EDP = DPELE(2)                                                  30
      BIDP = DPELE(3)
      HDP = DPELE(4)
      GDP = DPELE(5)
      BLDP = DPELE(6)
      A0 = ADP
      E0 = EDP
      BI0 = BIDP
      H0 = HDP
      G0 = GDP
      BL0 = BLDP                                                      40
      IFLG = 0
C---------------------I
C COMPUTE MEAN MOTION I
C---------------------I
      ANU=DSQRT(BMU/A0**3)
C--------------------I
C COMPUTE FRACTIONS I
C--------------------I
      F3D8=3.0D0/8.0D0
      F1d2=1.0D0/2.0D0                                                50
      F3D2=3.0D0/2.0D0
      F1D4=1.0D0/4.0D0
      F5D4=5.0D0/4.0D0
```

```
      F1D8=1.0D0/8.0D0
      F5D12=5.0D0/12.0D0
      F1D16=1.0D0/16.0D0
      F15D16=15.0D0/16.0D0
      F5D24=5.0D0/24.0D0
      F3D32=3.0D0/32.0D0
      F15D32=15.0D0/32.0D0                                              60
      F5D64=5.0D0/64.0D0
      F35384=35.0D0/384.0D0
      F35576=35.0D0/576.0D0
      F35D52=35.0D0/1152.0D0
      F1D3=1.0D0/3.0D0
      F5D16=5.0D0/16.0D0
      BK2 = -F1D2*(BJ2*RE*RE)
      BK3 = BJ3*RE**3
      BK4 = F3D8*(BJ4*RE**4)
      BK5 = BJ5*RE**5                                                   70
      GO TO 153
  111 IF(IPERT.EQ.0)GO TO 7
      IF(IDMEAN.NE.0)GO TO 202
      ADP = DPELE(1)/AE
      EDP = DPELE(2)
      BIDP = DPELE(3)
      HDP = DPELE(4)
      GDP = DPELE(5)
      BLDP = DPELE(6)
  153 EDP2=EDP*EDP                                                      80
      CN2=1.0-EDP2
      CN=DSQRT(CN2)
      GM2=BK2/ADP**2
      GMP2=GM2/(CN2*CN2)
      GM4=BK4/ADP**4
      GMP4=GM4/CN**8
      THETA=DCOS(BIDP)
      THETA2=THETA*THETA
C     A1DASH=1-5*THETA2
      A1DASH=(1-5*THETA2)/(1-EXP(-100*((1-5*THETA2)**2)))               90
      THETA4=THETA2*THETA2
  202 IF(IDMEAN.EQ.0)GO TO 155
      IF ( IPASS.EQ.2 ) GO TO 150
C-----------------------I
C COMPUTE LDOT,GDOT,HDOT I
C-----------------------I
  157 BLDOT=CN*ANU*(GMP2*(F3D2*(3.0*THETA2-1)+GMP2*F3D32*(THETA2
     1*(-96.0*CN+30.0-90.0*CN2)+(16.0*CN+25.0*CN2-15.0)+THETA4
     2*(144.0*CN+25.0*CN2+105.0)))+EDP2*GMP4*F15D16*(3.0+35.0*THETA4
     3-30.0*THETA2))                                                   100
      GDOT=ANU*(F5D16*GMP4*((THETA2*(126.0*CN2-270.0)+THETA4*(385.0
     1-189.0*CN2))-9.0*CN2+21.0)+GMP2*(F3D32*GMP2*(THETA4*(45.0*CN2
     2+360.0*CN+385.0)+THETA2*(90.0-192.0*CN-126.0*CN2)+(24.0*CN
     3+25.0*CN2-35))+F3D2*(5*THETA2-1)))
      HDOT=ANU*(GMP4*F5D4*THETA*(3.0-7.0*THETA2)*(5.0-3.0*CN2)+GMP2
     1*(GMP2*F3D8*(THETA*(12.0*CN+9.0*CN2-5.0)-THETA*THETA2*(5.0*CN2
     2+36.0*CN+35.0))-3*THETA))
```

```
  155 IF(IFLG.EQ.1)GO TO 19
CI-----------------------------------------I
CI COMPUTE ISUBC TO TEST CRITICAL INCLINATION I                    110
CI-----------------------------------------I
      BISUBC=((A1DASH)**(-2))*((25.0*THETA4*THETA)*(GMP2*EDP2))
      IFLG=1
CI-------------------------------------I
CI FIRST CHECK FOR CRITICAL INCLINATION I
CI-------------------------------------I
      IF(BISUBC.GT.BKSUBC)GO TO 158
      ASSIGN 163 TO ID8
      GO TO 159
C-------------------------------I                                  120
C IS THERE CRITICAL INCLINATION I
C-------------------------------I
   19 IF(BISUBC.GT.BKSUBC)GO TO 150
  159 IF(IPERT.EQ.1)GO TO 150
      GM3=BK3/ADP**3
      GMP3=GM3/(CN2*CN2*CN2)
      GM5=BK5/ADP**5
      GMP5=GM5/CN**10
      G3DG2=GMP3/GMP2
      G4DG2=GMP4/GMP2                                              130
      G5DG2=GMP5/GMP2
CI---------------I
CI COMPUTE A1-A8 I
CI---------------I
      A1=(F1D8*GMP2*CN2)*(1.0-11.0*THETA2-((40.0*THETA4)/(A1DASH
     1)))
      A2=(F5D12*G4DG2*CN2)*(1.0-((8.0*THETA4)/(A1DASH))-3.0
     1*THETA2)
      A3=G5DG2*((3.0*EDP2)+4.0)
      A4=G5DG2*(1.0-(24.0*THETA4)/(A1DASH)-9.0*THETA2)             140
      A5=(G5DG2*(3.0*EDP2+4.0))*(1.0-(24.0*THETA4)/(A1DASH)-9.0
     1*THETA2)
      A6=G3DG2*F1D4
      SINI=DSIN(BIDP)
      A10=CN2*SINI
      A7=A6*A10
      A8P=G5DG2*EDP*(1.0-(16.0*THETA4)/(A1DASH)-5.0*THETA2)
      A8=A8P*EDP
C
C     COMPUTE B13-B15                                              150
C
      B13=EDP*(A1-A2)
      B14=A7+F5D64*A5*A10
      B15=A8*A10*F35384
C
C    COMPUTE A11-A27
C
      A11=2.0+EDP2
      A12=3.0*EDP2+2.0
      A13=THETA2*A12                                               160
      A14=(5.0*EDP2+2.0)*(THETA4/(A1DASH))
```

```
      A17=THETA4/((A1DASH)*(A1DASH))
      A15=(EDP2*THETA4*THETA2)/((A1DASH)*(A1DASH))
      A16=THETA2/(A1DASH)
      A18=EDP*SINI
      A19=A18/(1.0+CN)
      A21=EDP*THETA
      A22=EDP2*THETA
      SINI2=DSIN(BIDP/2.0)
      COSI2=DCOS(BIDP/2.0)                                              170
      TANI2=DTAN(BIDP/2.0)
      A26=16.0*A16+40.0*A17+3.0
      A27=A22*F1D8*(11.0+200.0*A17+80.0*A16)
CI-----------------I
CI COMPUTE B1-B12 I
CI-----------------I
      B1=CN*(A1-A2)-((A11-400.0*A15-40.0*A14-11.0*A13)*F1D16+(11.0+200.0
     1*A17+80.0*A16)*A22*F1D8)*GMP2+((-80.0*A15-8.0*A14-3.0*A13+A11)
     2*F5D24+F5D12*A26*A22)*G4DG2
      B2=A6*A19*(2.0+CN-EDP2)+F5D64*A5*A19*CN2-F15D32*A4*A18*CN*CN2     180
     1+(F5D64*A5+A6)*A21*TANI2+(9.0*EDP2+26.0)*F5D64*A4*A18+F15D32*A3
     2*A21*A26*SINI*(1.0-THETA)
      B3=((80.0*A17+5.0+32.0*A16)*A22*SINI*(THETA-1.0)*F35576*G5DG2*EDP)
     1-((A22*TANI2+(2.0*EDP2+3.0*(1.0-CN2*CN))*SINI)*F35D52*A8P)
      B4=CN*EDP*(A1-A2)
      B5=((9.0*EDP2+4.0)*A10*A4*F5D64+A7)*CN
      B6=F35384*A8*CN2*CN*SINI
      B7=((CN2*A18)/(A1DASH))*(F1D8*GMP2*(1.0-15.0*THETA2)+(1.0
     1-7.0*THETA2)*G4DG2*(-F5D12))
      B8=F5D64*(A3*CN2*(1.0-9.0*THETA2-(24.0*THETA4/(A1DASH))))         190
     1+A6*CN2
      B9=A8*F35384*CN2
      B10=SINI*(A22*A26*G4DG2*F5D12-A27*GMP2)
      B11=A21*(A5*F5D64+A6+A3*A26*F15D32*SINI*SINI)
      B12=-((80.0*A17+32.0*A16+5.0)*(A22*EDP*SINI*SINI*F35576*G5DG2)+(A8
     1*A21*F35D52))
  150 IF(IPERT.EQ.0)GO TO 7
      IF(IDMEAN.EQ.0)GO TO 4
C-----------------------I
C COMPUTE SECULAR TERMS I                                               200
C-----------------------I
CI-----------------------I
CI ''MEAN'' MEAN ANOMALY I
CI ADD EFFECTS OF N(P,Q) I
CI-----------------------I
CDWC
CDWC  CALL BLDRAG(DRAGL)
CDWC  BLDP = ANU*DELT + BLDOT * DELT + BL0 + DRAGL
      BLDP = ANU*DELT + BLDOT * DELT + BL0
CDWC                                                                    210
      BLDP = DMOD(BLDP,2.0D0*PI)
      IF(BLDP.LT.0.0D0)BLDP = BLDP + PI*2.0D0
CI--------------------------I
CI  MEAN ARGUMENT OF PERIGEE I
CI--------------------------I
```

```
      GDP = GDOT*DELT  + G0
      GDP = DMOD(GDP,PI*2.0D0)
      IF(GDP.LT.0.0D0)GDP = GDP + PI*2.0D0
C     MEAN LONGITUDE OF ASCENDING NODE
      HDP = HDOT*DELT + H0                                              220
      HDP = DMOD(HDP,PI*2.0D0)
      IF(HDP.LT.0.0D0)HDP = HDP + PI*2.0D0
    4 DO 33 NN=1,6
   33 OSCELE(NN) = DPELE(NN)
      A = ADP
      E = EDP
      BI = BIDP
      H = HDP
      G = GDP
      BL = BLDP                                                         230
CI-----------------------------------I
CI COMPUTE TRUE ANOMALY(DOUBLE PRIMED) I
CI-----------------------------------I
      EADP = DKEPLR(BLDP,EDP,PI)
      SINDE= DSIN(EADP)
      COSDE= DCOS(EADP)
      SINFD= CN*SINDE
      COSFD= COSDE - EDP
CDWC
CDWC  FDP = DATAN0(SINFD,COSFD)                                         240
      FDP = DATAN2(SINFD,COSFD)
      IF(FDP.LT.0.D0) FDP=FDP+PI*2.0D0
CDWC
      IF(IPERT.EQ.1)GO TO 7
      DADR=(1.0-EDP*COSDE)**(-1)
      SINFD=SINFD*DADR
      COSFD=COSFD*DADR
      CS2GFD=DCOS(2.0*GDP+2.0*FDP)
      DADR2=DADR*DADR
      DADR3=DADR2*DADR                                                  250
      COSFD2=COSFD*COSFD
CI-----------------------------I
CI COMPUTE A(SEMI-MAJOR AXIS) I
CI-----------------------------I
      A=ADP*(1.0+GM2*((3.0*THETA2-1.0)*(EDP2/(CN2*CN2*CN2))*(CN+(1.0/(1.
     1+CN)))+((3.0*THETA2-1.0)/(CN2*CN2*CN2))*(EDP*COSFD)*(3.0+3.0*EDP
     2*COSFD+EDP2*COSFD2)+3.0*(1.0-THETA2)*DADR3*CS2GFD))
      SN2GFD=DSIN(2.0*GDP+2.0*FDP)
      SNF2GD=DSIN(2.0*GDP+FDP)
      CSF2GD=DCOS(2.0*GDP+FDP)                                          260
      SN2GD=DSIN(2.0*GDP)
      CS2GD=DCOS(2.0*GDP)
      SN3GD=DSIN(3.0*GDP)
      CS3GD=DCOS(3.0*GDP)
      SN3FGD=DSIN(3.0*FDP+2.0*GDP)
      CS3FGD=DCOS(3.0*FDP+2.0*GDP)
      SINGD=DSIN(GDP)
      COSGD=DCOS(GDP)
      GO TO ID8,(163,164)
```

```
 163 DLT1E=B14*SINGD+B13*CS2GD-B15*SN3GD                                  270
CI-----------------------I
CI COMPUTE (L+G+H) PRIMED I
CI-----------------------I
     BLGHP=HDP+GDP+BLDP+B3*CS3GD+B1*SN2GD+B2*COSGD
     BLGHP=DMOD(BLGHP,PI*2.0D0)
     IF(BLGHP.LT.0.0D0)BLGHP=BLGHP+PI*2.0D0
     EDPDL=B4*SN2GD-B5*COSGD+B6*CS3GD-F1D4*CN2*CN*GMP2*(2.0*(3.0*THETA2
    1-1.0)*(DADR2*CN2+DADR+1.0)*SINFD+3.0*(1.0-THETA2)*((-DADR2*CN2
    2-DADR+1.0)*SNF2GD+(DADR2*CN2+DADR+F1D3)*SN3FGD))
     DLTI=F1D2*THETA*GMP2*SINI*(EDP*CS3FGD+3.0*(EDP*CSF2GD+CS2GFD))      280
    1-(A21/CN2)*(B8*SINGD+B7*CS2GD-B9*SN3GD)
     SINDH=(1.0/COSI2)*(F1D2*(B12*CS3GD+B11*COSGD+B10*SN2GD-(F1D2*GMP2
    1*THETA*SINI*(6.0*(EDP*SINFD-BLDP+FDP)-(3.0*(SN2GFD+EDP*SNF2GD)+EDP
    2*SN3FGD)))))
CI------------------I
CI COMPUTE (L+G+H) I
CI------------------I
 164 BLGH=BLGHP+((1.0/(CN+1.0))*F1D4*EDP*GMP2*CN2*(3.0*(1.0-THETA2)*
    1(SN3FGD*(F1D3+DADR2*CN2+DADR)+SNF2GD*(1.0-(DADR2*CN2+DADR)))+2.0*
    2SINFD*(3.0*THETA2-1.0)*(DADR2*CN2+DADR+1.0)))+GMP2*F3D2*((-2.0*     290
    3THETA-1.0+5.0*THETA2)*(EDP*SINFD+FDP-BLDP))+(3.0+2.0*THETA-5.0*
    4THETA2)*(GMP2*F1D4*(EDP*SN3FGD+3.0*(SN2GFD+EDP*SNF2GD)))
     BLGH=DMOD(BLGH,PI*2.0D0)
     IF(BLGH.LT.0.0D0)BLGH=BLGH+PI*2.0D0
     DLTE=DLT1E+(F1D2*CN2*((3.0*(1.0/(CN2*CN2*CN2))*GM2*(1.0-THETA2)
    1*CS2GFD*(3.0*EDP*COSFD2+3.0*COSFD+EDP2*COSFD*COSFD2+EDP))-(GMP2
    2*(1.0-THETA2)*(3.0*CSF2GD+CS3FGD))+(3.0*THETA2-1.0)*GM2*(1.0/
    3(CN2*CN2*CN2))*(EDP*CN+(EDP/(1.0+CN))+3.0*EDP*COSFD2+3.0*COSFD+
    4EDP2*COSFD*COSFD2)))
     EDPDL2=EDPDL*EDPDL                                                  300
     EDPDE2=(EDP+DLTE)*(EDP+DLTE)
CI-------------------------I
CI COMPUTE E(ECCENTRICITY) I
CI-------------------------I
     E=DSQRT(EDPDL2+EDPDE2)
     if(e.ge.1.0d0)then
     intgr = 1
     write(1,*)'Failure in brolyd due to eccentricity > 1'
     goto 1000
     endif                                                               310
     SINDH2=SINDH*SINDH
     SQUAR=(DLTI*COSI2*F1D2+SINI2)*(DLTI*COSI2*F1D2+SINI2)
     SQRI=DSQRT(SINDH2+SQUAR)
     if(sqri.ge.1.0d0)then
     intgr = 1
     write(1,*)'Failure in brolyd due to problem with inclination'
     goto 1000
     endif
CI--------------------------I
CI COMPUTE BI (INCLINATION) I                                            320
CI--------------------------I
CDWC
CDWC  BI=DARSIN(SQRI)
```

```
      BI=  ASIN(SQRI)
CDWC
      BI=2.0*BI
      BI=DMOD(BI,PI*2.0D0)
      IF(BI.LT.0.0D0)BI=BI+PI*2.0D0
CI-----------------------------I
CI CHECK FOR E(ECCENTRICITY)=0 I                                        330
CI-----------------------------I
      IF(E.NE.0.0) GO TO 168
      BL=0.0
CI-----------------------------I
CI CHECK FOR BI(INCLINATION)=0 I
CI-----------------------------I
  145 IF(BI.NE.0.0) GO TO 169
      H=0.0
CI-----------------------------------I
CI COMPUTE G(ARGUMENT  OF PERIGEE) I                                    340
CI-----------------------------------I
  146 G=BLGH-BL-H
      G=DMOD(G,PI*2.0D0)
      IF(G.LT.0.0D0)G=G+PI*2.0D0
CI----------------------I
CI COMPUTE TRUE ANOMALY I
CI----------------------I
      EA   = DKEPLR(BL,E,PI)
      ARG1 = DSIN(EA) * DSQRT(1.0-E**2)
      ARG2 = DCOS(EA) - E                                               350
CDWC
CDWC  F = DATAN0(ARG1,ARG2)
      F = DATAN2(ARG1,ARG2)
      IF(F.LT.0.D0) F=F+PI*2.0D0
CDWC
C
C
      OSCELE(1) = A*AE
      OSCELE(2) = E
      OSCELE(3) = BI                                                    360
      OSCELE(4) = H
      OSCELE(5) = G
      OSCELE(6) = BL
    7 CONTINUE
      DPELE(1) = ADP*AE
      DPELE(2) = EDP
      DPELE(3) = BIDP
      DPELE(4) = HDP
      DPELE(5) = GDP
      DPELE(6) = BLDP+anu*delt                                          370
      IF(IPERT.EQ.0)BL = DMOD(ANU*DELT,PI*2.0D0)
      ORBEL(1) = EADP
      ORBEL(2) = GDP + FDP
      ORBEL(3) = GDP
      ORBEL(4) = EK*(ANU + BLDOT)
      ORBEL(5) = FDP
CDWC
```

```
CDWC  R = A*AE*(1.0D0 - E*DCOS(EA))
CDWC
      GO TO 45                                                          380
CI-----------------------------------------I
CI MODIFICATIONS FOR CRITICAL INCLINATION I
CI-----------------------------------------I
  158 DLT1E=0.0
      BLGHP=0.0
      EDPDL=0.0
      DLTI=0.0
      SINDH=0.0
      ASSIGN 164 TO ID8
      GO TO 150                                                         390
  168 SINLDP=DSIN(BLDP)
      COSLDP=DCOS(BLDP)
      SINHDP=DSIN(HDP)
      COSHDP=DCOS(HDP)
CI--------------------------I
CI COMPUTE L(MEAN ANOMALY) I
CI--------------------------I
      ARG1=EDPDL*COSLDP+(EDP+DLTE)*SINLDP
      ARG2=(EDP+DLTE)*COSLDP-(EDPDL*SINLDP)
      BL=DATAN2(ARG1,ARG2)                                              400
      BL=DMOD(BL,PI*2.0D0)
      IF(BL.LT.0.0D0)BL=BL+PI*2.0D0
      GO TO 145
CI-----------------------------------------I
CI COMPUTE H(LONGITUDE OF ASCENDING NODE) I
CI-----------------------------------------I
  169 ARG1=SINDH*COSHDP+SINHDP*(F1D2*DLTI*COSI2+SINI2)
      ARG2=COSHDP*(F1D2*DLTI*COSI2+SINI2)-(SINDH*SINHDP)
      H=DATAN2(ARG1,ARG2)
      H=DMOD(H,PI*2.0D0)                                                410
      IF(H.LT.0.0D0)H=H+PI*2.0D0
      GO TO 146
   45 CONTINUE
      RETURN
 1000 END

      DOUBLE PRECISION FUNCTION DKEPLR (M,E,PI)
C
C       SUBROUTINE TO SOLVE KEPLER'S EQ.
C  KEPLER'S EQ.,RELATES GEOMETRY OR POSITION IN ORBIT PLANE TO TIME.      420
C
C M - MEAN ANOMALY (0<M<2*PI)
C E - ECCENTRICITY
C EA- ECCENTRIC ANOMALY
C
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      DOUBLE PRECISION M
      TOL = 0.5D-17
      EA=0
      IF(M)1,2,1                                                        430
    1 EA=M + E*DSIN(M)
```

```
      DO 22 I=1,12
      OLDEA=EA
      FE=EA-E*DSIN(EA)-M
      EA=EA-FE/(1-E*DCOS(EA-0.5D0*FE))
C TEST FOR CONVERGENCE
      DELEA=DABS(EA-OLDEA)
      IF(DELEA.LE.TOL)GO TO 2
   22 CONTINUE
    2 EA=DMOD(EA,PI*2.0D0)                                          440
      DKEPLR=EA
      RETURN
      END
```

```
      SUBROUTINE  EQNKEP  (EQNELM,RETRO,  KEPELM,  SETRTR)
C
C
C
C           ********
C           FUNCTION
C           ********
C
C
C                                                                       10
C     THIS SUBROUTINE CONVERTS KEPLERIAN ORBIT ELEMENTS TO EQUINOCTIAL
C     ORBIT ELEMENTS.
C
C
C
C     CALLING SEQUENCE ***************************************************
C
C        CALL  EQNKEP  (EQNELM,RETRO,  KEPELM,  SETRTR)
C
C     PARAMETERS *********************************************************  20
C
C        EQNELM  O   EQUINOCTIAL ORBIT ELEMENTS.
C        RETRO  I/O  RETROGRADE FACTOR FOR EQUINOCTIAL ORBIT
C                    ELEMENTS.
C                     1  DIRECT ELEMENTS
C                    -1  RETROGRADE ELEMENTS
C
C        KEPELM  I   KEPLERIAN ORBIT ELEMENTS.
C
C        SETRTR  I   COMPUTE NEW VALUE OF RETROGRADE FACTOR FOR         30
C                    EQUINOCTIAL ORBIT ELEMENTS?
C                    .TRUE.   YES
C                    .FALSE.  NO
C
C     NO SUBROUTINES CALLED **********************************************
C
C
C
C**************** HISTORY ***********************************************
C                                                                       40
C
C
C     VERSION OF DECEMBER 1983
C        FORTRAN SUBROUTINE FOR THE AMDAHL 470/V8 AND THE IBM 3033.
C
C     ANALYSIS
C        LEO W. EARLY, JR.       -- CHARLES STARK DRAPER LABORATORY
C
C     PROGRAMMER
C        LEO W. EARLY, JR.       -- CHARLES STARK DRAPER LABORATORY      50
C
C
C
```

```
C***************** DECLARATIONS ***************************************
C
C
C
      IMPLICIT            DOUBLE PRECISION (A-H,O-Z)
      LOGICAL             SETRTR
      DOUBLE PRECISION    EQNELM (6)       ,KEPELM (6)                    60
      DOUBLE PRECISION    INCLIN           ,NODE
C
C
C
C***************** DATA STATEMENTS ************************************
C
C
C
      DATA HALFPI    /  1.5707 96326 79489 66   D 0  /
      DATA TWOPI     /  6.2831 85307 17958 65   D 0  /                    70
C
C
C
C***************** BEGIN PROGRAM **************************************
C
C
C
C           ************************
C           KEPLERIAN ORBIT ELEMENTS
C           ************************                                      80
C
C
C
      A      =  KEPELM (1)
      ECCEN  =  KEPELM (2)
      INCLIN =  KEPELM (3)
      NODE   =  KEPELM (4)
      PERGEE =  KEPELM (5)
      ANOMLY =  KEPELM (6)
C                                                                         90
C
C
C           *************************
C           EQUINOCTIAL ORBIT ELEMENTS
C           *************************
C
C
C
C           RETROGRADE FACTOR.
C                                                                        100
      IF  (SETRTR)  THEN
        RETRO  =  1.D0
        IF  (INCLIN .GT. HALFPI)  RETRO  =  - 1.D0
      END IF
C
C           AUXILIARY PARAMETERS.
C
```

```
      IF  (RETRO .GE. 0.D0)  THEN
        APPLAN  =  PERGEE + NODE
        TANINC  =  TAN (INCLIN/2.D0)                                    110
      ELSE
        APPLAN  =  PERGEE - NODE
        TANINC  =  TAN (HALFPI - INCLIN/2.D0)
      END IF
C
C             H  AND  K
C
      XH  =  ECCEN * SIN (APPLAN)
      XK  =  ECCEN * COS (APPLAN)
C                                                                       120
C             P  AND  Q
C
      P  =  TANINC * SIN (NODE)
      Q  =  TANINC * COS (NODE)
C
C             MEAN LONGITUDE
C
      XLAMDA  =  ANOMLY + APPLAN
C
C                                                                       130
C
C             *************
C             RETURN OUTPUT
C             *************
C
C
C
C
      EQNELM (1)  =  A
      EQNELM (2)  =  XH
      EQNELM (3)  =  XK                                                 140
      EQNELM (4)  =  P
      EQNELM (5)  =  Q
      EQNELM (6)  =  XLAMDA
      RETURN
      END
```

```
      SUBROUTINE  CARTES  (POS,VEL,  EQNELM,RETRO,  GM)
C
C
C
C           ********
C           FUNCTION
C           ********
C
C
C                                                                       10
C     THIS SUBROUTINE CONVERTS EQUINOCTIAL ORBIT ELEMENTS TO POSITION
C     AND VELOCITY.
C
C
C
C     CALLING SEQUENCE ************************************************
C
C        CALL  CARTES  (POS,VEL,  EQNELM,RETRO,  GM)
C
C     PARAMETERS ******************************************************  20
C
C        POS    O   POSITION.
C        VEL    O   VELOCITY.
C
C        EQNELM I   EQUINOCTIAL ORBIT ELEMENTS.
C        RETRO  I   RETROGRADE FACTOR.
C                 1  DIRECT ELEMENTS
C                -1  RETROGRADE ELEMENTS
C
C        GM     I   GRAVITATIONAL CONSTANT OF CENTRAL BODY.             30
C
C     /SATELM/ ********************************************************
C
C     POSITION AND VELOCITY.
C
C        X      O   POSITION.
C        V      O   VELOCITY.
C
C     AUXILIARY PARAMETERS FOR POSITION AND VELOCITY.
C                                                                       40
C        XORB   O   POSITION IN ORBITAL REFERENCE FRAME.
C        VORB   O   VELOCITY IN ORBITAL REFERENCE FRAME.
C        RDA    O   RADIAL DISTANCE / SEMIMAJOR AXIS
C
C     SATELLITE LONGITUDE.
C
C        F      O   ECCENTRIC LONGITUDE.
C        SINF   O   SIN (F)
C        COSF   O   COS (F)
C                                                                       50
C     EQUINOCTIAL ORBIT ELEMENTS.
C
C        A      O   SEMIMAJOR AXIS:  A
```

```
C      XH     O   H
C      XK     O   K
C      P      O   P
C      Q      O   Q
C      XLAMDA O   MEAN LONGITUDE:  LAMBDA
C
C      RETRG  O   RETROGRADE FACTOR.                                    60
C               1  DIRECT ELEMENTS
C              -1  RETROGRADE ELEMENTS
C
C    AUXILIARY PARAMETERS FOR EQUINOCTIAL ORBIT ELEMENTS.
C
C      FAXIS  O   F AXIS OF ORBITAL REFERENCE FRAME.
C      GAXIS  O   G AXIS OF ORBITAL REFERENCE FRAME.
C      WAXIS  O   W AXIS OF ORBITAL REFERENCE FRAME.
C
C      XMU    O   GRAVITATIONAL CONSTANT OF CENTRAL BODY.               70
C      XMEAN  O   KEPLER MEAN MOTION:  N
C      XNA    O   N*A
C      XNA2   O   N*A*A
C
C      XHH    O   H*H
C      XKK    O   K*K
C      HHKK   O   H*H + K*K
C      ONEHK  O   1 - H*H - K*K
C      SRTHK  O   SQRT (1 - H*H - K*K)
C      BETAUX O   1 / (1 + SQRT (1 - H*H - K*K))                        80
C      ONEHHB O   1 - H*H*BETAUX
C      ONEKKB O   1 - K*K*BETAUX
C      HKBETA O   H*K*BETAUX
C
C      PP     O   P*P
C      QQ     O   Q*Q
C      PPQQ   O   P*P + Q*Q
C      ONEPQ  O   1 + P*P + Q*Q
C      ONEPQI O   1 / (1 + P*P + Q*Q)
C                                                                       90
C    SUBROUTINES CALLED **********************************************
C
C      AUXEQN   ECLONG
C
C
C
C**************** HISTORY ***************************************
C
C
C                                                                      100
C    VERSION OF FEBRUARY 1984
C       FORTRAN SUBROUTINE FOR THE IBM 3081 AND 3033.
C
C    ANALYSIS
C       LEO W. EARLY, JR.        -- CHARLES STARK DRAPER LABORATORY
C
C    PROGRAMMER
```

```
C       LEO W. EARLY, JR.       -- CHARLES STARK DRAPER LABORATORY
C
C                                                                       110
C
C***************** DECLARATIONS ***************************************
C
C
C
      IMPLICIT        DOUBLE PRECISION (A-H,O-Z)
C
C     DIMENSIONS ***********************************************************
C
      DIMENSION       EQNELM (6)          ,FAXIS (3)                    120
      DIMENSION       POS (3)             ,GAXIS (3)
      DIMENSION       VEL (3)             ,WAXIS (3)
      DIMENSION       X  (3)              ,XORB (2)
      DIMENSION       V  (3)              ,VORB (2)
C
C     /SATELM/ *************************************************************
C
      COMMON /SATELM/  RSATEL (87)
C
C             POSITION AND VELOCITY.                                    130
C
      EQUIVALENCE     (X     (1)          ,RSATEL (1)          ),
     *          (V     (1)          ,RSATEL (4)          )
C
C             AUXILIARY PARAMETERS FOR POSITION AND VELOCITY.
C
      EQUIVALENCE     (XORB  (1)          ,RSATEL (7)          ),
     *          (VORB  (1)          ,RSATEL (9)          ),
     *          (RDA                ,RSATEL (11)         )
C                                                                       140
C             SATELLITE LONGITUDE.
C
      EQUIVALENCE     (F                  ,RSATEL (12)         ),
     *          (SINF               ,RSATEL (13)         ),
     *          (COSF               ,RSATEL (14)         )
C
C             EQUINOCTIAL ORBIT ELEMENTS.
C
      EQUIVALENCE     (A                  ,RSATEL (15)         ),
     *          (XH                 ,RSATEL (16)         ),          150
     *          (XK                 ,RSATEL (17)         ),
     *          (P                  ,RSATEL (18)         ),
     *          (Q                  ,RSATEL (19)         ),
     *          (XLAMDA             ,RSATEL (20)         ),
     *          (RETRG              ,RSATEL (21)         )
C
C             AUXILIARY PARAMETERS FOR EQUINOCTIAL ORBIT ELEMENTS.
C
      EQUIVALENCE     (FAXIS (1)          ,RSATEL (22)         ),
     *          (GAXIS (1)          ,RSATEL (25)         ),          160
     *          (WAXIS (1)          ,RSATEL (28)         ),
```

```
     *           (XMU              ,RSATEL (31)        ),
     *           (XMEAN             ,RSATEL (32)        ),
     *           (XNA              ,RSATEL (33)        ),
     *           (XNA2             ,RSATEL (34)        ),
     *           (XHH              ,RSATEL (35)        ),
     *           (XKK              ,RSATEL (36)        ),
     *           (HHKK             ,RSATEL (37)        )
     EQUIVALENCE    (ONEHK              ,RSATEL (38)        ),
     *           (SRTHK            ,RSATEL (39)        ),                    170
     *           (BETAUX            ,RSATEL (40)        ),
     *           (ONEHHB            ,RSATEL (41)        ),
     *           (ONEKKB            ,RSATEL (42)        ),
     *           (HKBETA            ,RSATEL (43)        ),
     *           (PP               ,RSATEL (44)        ),
     *           (QQ               ,RSATEL (45)        ),
     *           (PPQQ              ,RSATEL (46)        ),
     *           (ONEPQ             ,RSATEL (47)        ),
     *           (ONEPQI            ,RSATEL (48)        )
C                                                                            180
C
C
C**************** BEGIN PROGRAM ***************************************
C
C
C
C          ********************
C          AUXILIARY PARAMETERS
C          ********************
C                                                                            190
C
C
C          AUXILIARY PARAMETERS FOR EQUINOCTIAL ELEMENTS.
C
     CALL  AUXEQN  (EQNELM,RETRO,  GM)
C
C          ECCENTRIC LONGITUDE.
C
     SINLAM  =  SIN (XLAMDA)
     COSLAM  =  COS (XLAMDA)                                                 200
     CALL  ECLONG  (F,SINF,COSF,  XLAMDA,SINLAM,COSLAM,  XH,XK)
C
C          AUXILIARY PARAMETERS FOR POSITION AND VELOCITY.
C
     RDA    =  1.D0  -  XH * SINF  -  XK * COSF
     XNA2DR  =  XNA / RDA
C
C
C
C          ********************                                             210
C          POSITION AND VELOCITY
C          ********************
C
C
C
```

```
C           POSITION IN EQUINOCTIAL REFERENCE FRAME.
C
      XORB (1)  =  A * (ONEHHB*COSF + HKBETA*SINF - XK)
      XORB (2)  =  A * (ONEKKB*SINF + HKBETA*COSF - XH)
C                                                                    220
C           VELOCITY IN EQUINOCTIAL REFERENCE FRAME.
C
      VORB (1)  =  XNA2DR * (HKBETA*COSF - ONEHHB*SINF)
      VORB (2)  =  XNA2DR * (ONEKKB*COSF - HKBETA*SINF)
C
C           POSITION IN INERTIAL REFERENCE FRAME.
C
      X (1)  =  XORB (1) * FAXIS (1)  +  XORB (2) * GAXIS (1)
      X (2)  =  XORB (1) * FAXIS (2)  +  XORB (2) * GAXIS (2)
      X (3)  =  XORB (1) * FAXIS (3)  +  XORB (2) * GAXIS (3)        230
C
C           VELOCITY IN INERTIAL REFERENCE FRAME.
C
      V (1)  =  VORB (1) * FAXIS (1)  +  VORB (2) * GAXIS (1)
      V (2)  =  VORB (1) * FAXIS (2)  +  VORB (2) * GAXIS (2)
      V (3)  =  VORB (1) * FAXIS (3)  +  VORB (2) * GAXIS (3)
C
C
C
C           *************                                            240
C           RETURN OUTPUT
C           *************
C
C
C
C           RETURN POSITION TO CALLING PROGRAM.
C
      POS (1)  =  X (1)
      POS (2)  =  X (2)
      POS (3)  =  X (3)                                              250
C
C           RETURN VELOCITY TO CALLING PROGRAM.
C
      VEL (1)  =  V (1)
      VEL (2)  =  V (2)
      VEL (3)  =  V (3)
      RETURN
      END
```

```
      SUBROUTINE  AUXEQN  (ELEMNT,RETRO,  GM)
C
C
C
C           ********
C           FUNCTION
C           ********
C
C                                                                       10
C
C     THIS SUBROUTINE COMPUTES AUXILIARY PARAMETERS RELATED TO THE
C     EQUINOCTIAL ORBIT ELEMENTS.
C
C
C
C     CALLING SEQUENCE ************************************************
C
C        CALL  AUXEQN  (ELEMNT,RETRO,  GM)
C                                                                       20
C     PARAMETERS ******************************************************
C
C        ELEMNT I   EQUINOCTIAL ORBIT ELEMENTS.
C        RETRO  I   RETROGRADE FACTOR.
C                    1  DIRECT ELEMENTS
C                   -1  RETROGRADE ELEMENTS
C
C        GM     I   GRAVITATIONAL CONSTANT OF CENTRAL BODY.
C
C     /SATELM/ ********************************************************  30
C
C     EQUINOCTIAL ORBIT ELEMENTS.
C
C        A      O  SEMIMAJOR AXIS:  A
C        XH     O  H
C        XK     O  K
C        P      O  P
C        Q      O  Q
C        XLAMDA O  MEAN LONGITUDE:  LAMBDA
C                                                                       40
C        RETRG  O   RETROGRADE FACTOR.
C                    1  DIRECT ELEMENTS
C                   -1  RETROGRADE ELEMENTS
C
C     AUXILIARY PARAMETERS FOR EQUINOCTIAL ORBIT ELEMENTS.
C
C        FAXIS  O   F AXIS OF ORBITAL REFERENCE FRAME.
C        GAXIS  O   G AXIS OF ORBITAL REFERENCE FRAME.
C        WAXIS  O   W AXIS OF ORBITAL REFERENCE FRAME.
C                                                                       50
C        XMU    O   GRAVITATIONAL CONSTANT OF CENTRAL BODY.
C        XMEAN  O   KEPLER MEAN MOTION:  N
C        XNA    O   N*A
```

```
C       XNA2   O   N*A*A
C
C       XHH    O   H*H
C       XKK    O   K*K
C       HHKK   O   H*H + K*K
C       ONEHK  O   1 - H*H - K*K
C       SRTHK  O   SQRT (1 - H*H - K*K)                                    60
C       BETAUX O   1 / (1 + SQRT (1 - H*H - K*K))
C       ONEHHB O   1 - H*H*BETAUX
C       ONEKKB O   1 - K*K*BETAUX
C       HKBETA O   H*K*BETAUX
C
C       PP     O   P*P
C       QQ     O   Q*Q
C       PPQQ   O   P*P + Q*Q
C       ONEPQ  O   1 + P*P + Q*Q
C       ONEPQI O   1 / (1 + P*P + Q*Q)                                     70
C
C   NO SUBROUTINES CALLED ******************************************
C
C
C
C***************** HISTORY *****************************************
C
C
C
C   VERSION OF FEBRUARY 1984                                               80
C      FORTRAN SUBROUTINE FOR THE IBM 3081 AND 3033.
C
C   ANALYSIS
C      LEO W. EARLY, JR.       -- CHARLES STARK DRAPER LABORATORY
C
C   PROGRAMMER
C      LEO W. EARLY, JR.       -- CHARLES STARK DRAPER LABORATORY
C
C
C                                                                          90
C***************** DECLARATIONS ************************************
C
C
C
   IMPLICIT       DOUBLE PRECISION (A-H,O-Z)
C
C   DIMENSIONS *************************************************
C
   DIMENSION      ELEMNT(6)  ,FAXIS(3)  ,GAXIS(3)  ,WAXIS(3)
C                                                                         100
C   /SATELM/ ***************************************************
C
   COMMON /SATELM/  RSATEL (87)
C
C          EQUINOCTIAL ORBIT ELEMENTS.
C
   EQUIVALENCE    (A              ,RSATEL (15)       ),
```

```
     *          (XH             ,RSATEL (16)       ),
     *          (XK             ,RSATEL (17)       ),
     *          (P              ,RSATEL (18)       ),                    110
     *          (Q              ,RSATEL (19)       ),
     *          (XLAMDA           ,RSATEL (20)       ),
     *          (RETRG            ,RSATEL (21)       )
C
C          AUXILIARY PARAMETERS FOR EQUINOCTIAL ORBIT ELEMENTS.
C
      EQUIVALENCE    (FAXIS (1)        ,RSATEL (22)       ),
     *          (GAXIS (1)        ,RSATEL (25)       ),
     *          (WAXIS (1)        ,RSATEL (28)       ),
     *          (XMU              ,RSATEL (31)       ),                    120
     *          (XMEAN            ,RSATEL (32)       ),
     *          (XNA              ,RSATEL (33)       ),
     *          (XNA2             ,RSATEL (34)       ),
     *          (XHH              ,RSATEL (35)       ),
     *          (XKK              ,RSATEL (36)       ),
     *          (HHKK             ,RSATEL (37)       )
      EQUIVALENCE    (ONEHK            ,RSATEL (38)       ),
     *          (SRTHK            ,RSATEL (39)       ),
     *          (BETAUX           ,RSATEL (40)       ),
     *          (ONEHHB           ,RSATEL (41)       ),                    130
     *          (ONEKKB           ,RSATEL (42)       ),
     *          (HKBETA           ,RSATEL (43)       ),
     *          (PP             ,RSATEL (44)       ),
     *          (QQ             ,RSATEL (45)       ),
     *          (PPQQ             ,RSATEL (46)       ),
     *          (ONEPQ            ,RSATEL (47)       ),
     *          (ONEPQI           ,RSATEL (48)       )
C
C
C                                                                         140
C***************** BEGIN PROGRAM ************************************
C
C
C
C          EQUINOCTIAL ELEMENTS.
C
      A      =  ELEMNT (1)
      XH     =  ELEMNT (2)
      XK     =  ELEMNT (3)
      P      =  ELEMNT (4)                                                150
      Q      =  ELEMNT (5)
      XLAMDA =  ELEMNT (6)
      RETRG  =  RETRO
C
C          AUXILIARY KEPLER MEAN MOTION PARAMETERS.
C
      XMU    =  GM
      XMEAN  =  SQRT (XMU / (A*A*A))
      XNA    =  XMEAN * A
      XNA2   =  XNA * A                                                   160
C
```

```
C           AUXILIARY H AND K PARAMETERS.
C
      XHH    =  XH * XH
      XKK    =  XK * XK
      HHKK   =  XHH  +  XKK
      ONEHK  =  1.D0  -  HHKK
      SRTHK  =  SQRT (ONEHK)
      BETAUX =  1.D0 / (1.D0 + SRTHK)
      ONEHHB =  1.D0  -  XHH * BETAUX                                    170
      ONEKKB =  1.D0  -  XKK * BETAUX
      HKBETA =  XH * XK * BETAUX
C
C           AUXILIARY P AND Q PARAMETERS.
C
      PP     =  P * P
      QQ     =  Q * Q
      PPQQ   =  PP  +  QQ
      ONEPQ  =  1.D0  +  PPQQ
      ONEPQI =  1.D0 / ONEPQ                                             180
C
C           ORBITAL REFERENCE FRAME.
C
      FAXIS (1)  =  (1.D0 - PP + QQ) * ONEPQI
      FAXIS (2)  =        2.D0*P*Q * ONEPQI
      FAXIS (3)  =        - 2.D0*P * ONEPQI * RETRG
C
      GAXIS (1)  =        2.D0*P*Q * ONEPQI * RETRG
      GAXIS (2)  =  (1.D0 + PP - QQ) * ONEPQI * RETRG
      GAXIS (3)  =          2.D0*Q * ONEPQI                              190
C
      WAXIS (1)  =          2.D0*P * ONEPQI
      WAXIS (2)  =        - 2.D0*Q * ONEPQI
      WAXIS (3)  =    (1.D0 - PPQQ) * ONEPQI * RETRG
      RETURN
      END
```

```
      SUBROUTINE  ECLONG  (F,SINF,COSF,  XLAMDA,SINLAM,COSLAM,
     *                XH,XK)
C
C
C
C           ********
C           FUNCTION
C           ********
C
C                                                                       10
C
C     This subroutine solves Kepler's equation
C
C           LAMBDA   =   F   +   h cos F   -   k sin F
C
C       for the eccentric longitude   F   using Newton's method for
C     solving nonlinear equations.
C
C
C                                                                       20
C     CALLING SEQUENCE ************************************************
C
C        CALL  ECLONG  (F,SINF,COSF,  XLAMDA,SINLAM,COSLAM,
C       *           XH,XK)
C
C     PARAMETERS ******************************************************
C
C        F      O   Eccentric longitude.
C        SINF   O   sin F
C        COSF   O   cos F                                               30
C
C        XLAMDA I   Mean longitude.
C        SINLAM I   sin XLAMDA
C        COSLAM I   cos XLAMDA
C
C        XH     I   Equinoctial element h.
C        XK     I   Equinoctial element k.
C
C     NO SUBROUTINES CALLED *******************************************
C                                                                       40
C
C
C     RESTRICTIONS ****************************************************
C
C        SQRT (XH**2 + XK**2)     LE     .99999 99999
C
C
C
C***************** HISTORY ********************************************
C                                                                       50
C
C
C     VERSION:  January 1986
```

```
C       Fortran subroutine for the IBM 3081 and 3033.
C
C    ANALYSIS
C       Leo W. Early, Jr.          -- Charles Stark Draper Laboratory
C
C    PROGRAMMER
C       Leo W. Early, Jr.          -- Charles Stark Draper Laboratory                60
C
C
C
C**************** DECLARATIONS ***************************************
C
C
C
   IMPLICIT       DOUBLE PRECISION (A-H,O-Z)
C
C                                                                                    70
C
C**************** DATA STATEMENTS ************************************
C
C
C
C           Convergence tolerance.  The value given guarantees
C           that F, sin F, and cos F will have the maximum
C           accuracy obtainable with 16.8-digit floating-point
C           arithmetic.
C                                                                                    80
   DATA TOLER     /  1. D -10  /
C
C           Iteration limit.
C
   DATA ITRLIM    / 30 /
C
C           Error message file number.
C
   DATA MSG       / 6 /
C                                                                                    90
C
C
C**************** BEGIN PROGRAM **************************************
C
C
C
C           **********
C           INITIALIZE
C           **********
C                                                                                   100
C
C
C           Compute auxiliary parameters.
C
   R  =  XH * COSLAM  -  XK * SINLAM
   S  =  XH * SINLAM  +  XK * COSLAM
C
```

```
C            Compute initial approximation for  F - LAMBDA.
C
      FSL    =  0.D0                                                    110
      DELFSL =  R / (1.D0 - S)
      IF  (ABS (DELFSL) .GT. 1.D0)  DELFSL  =  SIGN (1.D0,DELFSL)
C
C
C
C            *******
C            ITERATE
C            *******
C
C                                                                       120
C
C            Compute next approximation for  F - LAMBDA.
C
      DO 100 ITR = 1,ITRLIM
        FSL    =  FSL - DELFSL
        SINFSL =  SIN (FSL)
        COSFSL =  COS (FSL)
C
C            Compute correction.
C                                                                       130
        DELFSL =  (FSL + R*COSFSL - S*SINFSL) / (1.D0 - R*SINFSL -
     *       S*COSFSL)
        IF  (ABS (DELFSL) .LE. TOLER)  GO TO 200
  100 CONTINUE
C
C
C
C            **************************
C            ITERATION DID NOT CONVERGE
C            **************************                                140
C
C
C
      WRITE  (MSG,2000)  ITRLIM,  XLAMDA,SINLAM,COSLAM,XH,XK,
     *            DELFSL
C
C
C
C            ****************
C            END OF ALGORITHM                                           150
C            ****************
C
C
C
C            Compute final approximation to  F - LAMBDA.
C
  200 DIF    =  FSL - DELFSL
      SINDIF =  SINFSL  -  COSFSL * DELFSL
      COSDIF =  COSFSL  +  SINFSL * DELFSL
C                                                                       160
C            Compute  F, sin F, cos F.
```

```
C
      F    =  XLAMDA + DIF
      SINF  =  SINLAM * COSDIF  +  COSLAM * SINDIF
      COSF  =  COSLAM * COSDIF  -  SINLAM * SINDIF
      RETURN
C
C
C
C***************** FORMAT STATEMENTS ************************************      170
C
C
C
 2000 FORMAT( /// ' ***** ERROR *****     ITERATION TO SOLVE KEPLER''',
     *     'S EQUATION FOR THE ECCENTRIC LONGITUDE FAILED TO' /
     *     ' CONVERGE IN ',I4,' ITERATIONS.' //
     *     ' LAMBDA = ',1PG13.5 /
     *     ' SINLAM = ',1PG13.5 /
     *     ' COSLAM = ',1PG13.5 /
     *     ' H      = ',1PG13.5 /                                            180
     *     ' K      = ',1PG13.5 //
     *     ' LAST CHANGE IN F = ',1PG13.5 /// )
      END
```

```
      SUBROUTINE  KEPEQN  (KEPELM,  EQNELM,RETRO,x)
C
C
C
C           ********
C           FUNCTION
C           ********
C
C
C                                                                       10
C     THIS SUBROUTINE CONVERTS EQUINOCTIAL ORBIT ELEMENTS TO KEPLERIAN
C     ORBIT ELEMENTS.
C
C
C
C     CALLING SEQUENCE ************************************************
C
C        CALL  KEPEQN  (KEPELM,  EQNELM,RETRO)
C
C     PARAMETERS ******************************************************  20
C
C        KEPELM  O   KEPLERIAN ORBIT ELEMENTS.
C
C        EQNELM  I   EQUINOCTIAL ORBIT ELEMENTS.
C        RETRO   I   RETROGRADE FACTOR.
C                  1  DIRECT ELEMENTS
C                 -1  RETROGRADE ELEMENTS
C
C     NO SUBROUTINES CALLED *******************************************
C                                                                       30
C
C
C**************** HISTORY ***************************************
C
C
C
C     VERSION OF DECEMBER 1983
C        FORTRAN SUBROUTINE FOR THE IBM 3081 AND 3033.
C
C     ANALYSIS                                                          40
C        LEO W. EARLY, JR.       -- CHARLES STARK DRAPER LABORATORY
C
C     PROGRAMMER
C        LEO W. EARLY, JR.       -- CHARLES STARK DRAPER LABORATORY
C
C
C
C**************** DECLARATIONS ***************************************
C
C                                                                       50
C
      IMPLICIT            DOUBLE PRECISION (A-H,O-Z)
      DOUBLE PRECISION    EQNELM (6),x(6)    ,KEPELM (6)
```

```
      DOUBLE PRECISION      INCLIN          ,NODE
C
C
C
C***************** DATA STATEMENTS ************************************
C
C                                                                       60
C
      DATA PI       /  3.1415 92653 58979 32   D 0  /
      DATA TWOPI    /  6.2831 85307 17958 65   D 0  /
C
C
C
C***************** BEGIN PROGRAM **************************************
C
C
C                                                                       70
C           ********************
C           EQUINOCTIAL ELEMENTS
C           ********************
C
C
C
      A      =  EQNELM (1)
      XH     =  EQNELM (2)
      XK     =  EQNELM (3)
      P      =  EQNELM (4)                                              80
      Q      =  EQNELM (5)
      XLAMDA =  EQNELM (6)
C
C
C
C           ******************
C           KEPLERIAN ELEMENTS
C           ******************
C
C                                                                       90
C
C           AUXILIARY PARAMETERS
C
      HHKK  =  XH * XH  +  XK * XK
      PPQQ  =  P * P  +  Q * Q
C
C           ECCENTRICITY
C
      ECCEN  =  SQRT (HHKK)
C                                                                      100
C           INCLINATION
C
      INCLIN  =  2.D0 * ATAN (SQRT (PPQQ))
      IF  (RETRO .LT. 0.D0)  INCLIN  =  PI - INCLIN
 25   if(inclin-x(3).gt.pi)then
      inclin = inclin-twopi
      goto 25
```

```
      endif
 40   if(x(3)-inclin.gt.pi)then
      inclin = inclin+twopi                                          110
      goto 40
      endif
C
C             LONGITUDE OF ASCENDING NODE
C
      IF  (PPQQ .GT. 0.D0)  THEN
        NODE  =  ATAN2 (P,Q)
      ELSE
        NODE  =  0.D0
      END IF                                                         120
 50   if(node-x(4).gt.pi)then
      node = node-twopi
      goto 50
      endif
 70   if(x(4)-node.gt.pi)then
      node = node+twopi
      goto 70
      endif
C
C             LONGITUDE OF PERIGEE                                   130
C
      IF  (HHKK .GT. 0.D0)  THEN
        APPLAN  =  ATAN2 (XH,XK)
      ELSE
        APPLAN  =  RETRO * NODE
      END IF
C
C             ARGUMENT OF PERIGEE
C
      PERGEE  =  APPLAN - RETRO*NODE                                 140
 75   if(pergee-x(5).gt.pi)then
      pergee = pergee-twopi
      goto 75
      endif
 90   if(x(5)-pergee.gt.pi)then
      pergee = pergee+twopi
      goto 90
      endif
C
C             MEAN ANOMALY                                           150
C
      ANOMLY  =  XLAMDA - APPLAN
 100  if(anomly-x(6).gt.pi)then
      anomly = anomly-twopi
      goto 100
      endif
 120  if(x(6)-anomly.gt.pi)then
      anomly = anomly+twopi
      goto 120
      endif                                                          160
C
```

```
C
C
C          *************
C          RETURN OUTPUT
C          *************
C
C
C
     KEPELM (1)  =  A                                                  170
     KEPELM (2)  =  ECCEN
     KEPELM (3)  =  INCLIN
     KEPELM (4)  =  NODE
     KEPELM (5)  =  PERGEE
     KEPELM (6)  =  ANOMLY
     RETURN
     END
```

```
      SUBROUTINE  CRTDRV  (DPVDEQ)
C
C
C
C           ********
C           FUNCTION
C           ********
C
C
C                                                                       10
C     THIS SUBROUTINE COMPUTES THE PARTIAL DERIVATIVES OF POSITION AND
C     VELOCITY WITH RESPECT TO THE EQUINOCTIAL ORBIT ELEMENTS.
C
C
C
C     CALLING SEQUENCE ************************************************
C
C        CALL  CRTDRV  (DPVDEQ)
C
C     PARAMETERS ******************************************************  20
C
C        DPVDEQ  O   PARTIAL DERIVATIVES OF POSITION AND VELOCITY
C                    WITH RESPECT TO EQUINOCTIAL ORBIT ELEMENTS.
C
C                    FOR J = 1,2,3
C
C        (J,1)       D X(J)  /  D A
C        (J,2)       D X(J)  /  D H
C        (J,3)       D X(J)  /  D K
C        (J,4)       D X(J)  /  D P                                     30
C        (J,5)       D X(J)  /  D Q
C        (J,6)       D X(J)  /  D LAMBDA
C
C        (J+3,1)     D V(J)  /  D A
C        (J+3,2)     D V(J)  /  D H
C        (J+3,3)     D V(J)  /  D K
C        (J+3,4)     D V(J)  /  D P
C        (J+3,5)     D V(J)  /  D Q
C        (J+3,6)     D V(J)  /  D LAMBDA
C                                                                       40
C     /SATELM/ ********************************************************
C
C     POSITION AND VELOCITY.
C
C        X      I   POSITION.
C        V      I   VELOCITY.
C
C     AUXILIARY PARAMETERS FOR POSITION AND VELOCITY.
C
C        XORB   I   POSITION IN ORBITAL REFERENCE FRAME.                 50
C        VORB   I   VELOCITY IN ORBITAL REFERENCE FRAME.
C        RDA    I   RADIAL DISTANCE / SEMIMAJOR AXIS
C
```

```
C   SATELLITE LONGITUDE.
C
C      F     I  ECCENTRIC LONGITUDE.
C      SINF  I  SIN (F)
C      COSF  I  COS (F)
C
C   EQUINOCTIAL ORBIT ELEMENTS.                                        60
C
C      A     I  SEMIMAJOR AXIS
C      XH    I  H
C      XK    I  K
C      P     I  P
C      Q     I  Q
C
C      RETRG I  RETROGRADE FACTOR.
C               1  DIRECT ELEMENTS
C              -1  RETROGRADE ELEMENTS                                 70
C
C   AUXILIARY PARAMETERS FOR EQUINOCTIAL ORBIT ELEMENTS.
C
C      FAXIS  I  F AXIS OF ORBITAL REFERENCE FRAME.
C      GAXIS  I  G AXIS OF ORBITAL REFERENCE FRAME.
C      WAXIS  I  W AXIS OF ORBITAL REFERENCE FRAME.
C
C      XMU    I  GRAVITATIONAL CONSTANT OF CENTRAL BODY.
C      XMEAN  I  KEPLER MEAN MOTION:  N
C      XNA    I  N*A                                                  80
C      XNA2   I  N*A*A
C
C      XHH    I  H*H
C      XKK    I  K*K
C      HHKK   I  H*H + K*K
C      ONEHK  I  1 - H*H - K*K
C      SRTHK  I  SQRT (1 - H*H - K*K)
C      BETAUX I  1 / (1 + SQRT (1 - H*H - K*K))
C      ONEHHB I  1 - H*H*BETAUX
C      ONEKKB I  1 - K*K*BETAUX                                       90
C      HKBETA I  H*K*BETAUX
C
C      PP     I  P*P
C      QQ     I  Q*Q
C      PPQQ   I  P*P + Q*Q
C      ONEPQ  I  1 + P*P + Q*Q
C      ONEPQI I  1 / (1 + P*P + Q*Q)
C
C   NO SUBROUTINES CALLED ******************************************
C                                                                     100
C
C
C   RESTRICTIONS ***************************************************
C
C      CALL SUBROUTINE "CARTES" TO CREATE THE INPUT FOR THIS
C      SUBROUTINE.
C
```

```
C
C
C**************** HISTORY ***************************************** 110
C
C
C
C   VERSION OF FEBRUARY 1984
C      FORTRAN SUBROUTINE FOR THE IBM 3081 AND 3033.
C
C   ANALYSIS
C      LEO W. EARLY, JR.      -- CHARLES STARK DRAPER LABORATORY
C
C   PROGRAMMER                                                    120
C      LEO W. EARLY, JR.      -- CHARLES STARK DRAPER LABORATORY
C
C
C
C**************** DECLARATIONS ***************************************
C
C
C
      IMPLICIT       DOUBLE PRECISION (A-H,O-Z)
C                                                                   130
C     DIMENSIONS ******************************************************
C
      DIMENSION      X (3)              ,XORB (2)
      DIMENSION      V (3)              ,VORB (2)
      DIMENSION      FAXIS (3)           ,WAXIS (3)
      DIMENSION      GAXIS (3)           ,DPVDEQ (6,6)
C
C     /SATELM/ ********************************************************
C
      COMMON /SATELM/  RSATEL (87)                                  140
C
C              POSITION AND VELOCITY.
C
      EQUIVALENCE    (X     (1)          ,RSATEL (1)          ),
     *               (V     (1)          ,RSATEL (4)          )
C
C              AUXILIARY PARAMETERS FOR POSITION AND VELOCITY.
C
      EQUIVALENCE    (XORB  (1)          ,RSATEL (7)          ),
     *               (VORB  (1)          ,RSATEL (9)          ),   150
     *               (RDA                ,RSATEL (11)         )
C
C              SATELLITE LONGITUDE.
C
      EQUIVALENCE    (F                  ,RSATEL (12)         ),
     *               (SINF               ,RSATEL (13)         ),
     *               (COSF               ,RSATEL (14)         )
C
C              EQUINOCTIAL ORBIT ELEMENTS.
C                                                                   160
      EQUIVALENCE    (A                  ,RSATEL (15)         ),
```

```
     *           (XH              ,RSATEL (16)       ),
     *           (XK              ,RSATEL (17)       ),
     *           (P               ,RSATEL (18)       ),
     *           (Q               ,RSATEL (19)       ),
     *           (RETRG             ,RSATEL (21)       )
C
C          AUXILIARY PARAMETERS FOR EQUINOCTIAL ORBIT ELEMENTS.
C
      EQUIVALENCE    (FAXIS (1)        ,RSATEL (22)       ),                  170
     *           (GAXIS (1)        ,RSATEL (25)       ),
     *           (WAXIS (1)        ,RSATEL (28)       ),
     *           (XMU              ,RSATEL (31)       ),
     *           (XMEAN              ,RSATEL (32)       ),
     *           (XNA              ,RSATEL (33)       ),
     *           (XNA2             ,RSATEL (34)       ),
     *           (XHH              ,RSATEL (35)       ),
     *           (XKK              ,RSATEL (36)       ),
     *           (HHKK               ,RSATEL (37)       )
      EQUIVALENCE    (ONEHK              ,RSATEL (38)       ),                  180
     *           (SRTHK            ,RSATEL (39)       ),
     *           (BETAUX             ,RSATEL (40)       ),
     *           (ONEHHB             ,RSATEL (41)       ),
     *           (ONEKKB             ,RSATEL (42)       ),
     *           (HKBETA             ,RSATEL (43)       ),
     *           (PP               ,RSATEL (44)       ),
     *           (QQ               ,RSATEL (45)       ),
     *           (PPQQ               ,RSATEL (46)       ),
     *           (ONEPQ              ,RSATEL (47)       ),
     *           (ONEPQI             ,RSATEL (48)       )                   190
C
C
C
C**************** BEGIN PROGRAM ****************************************
C
C
C
C          ********************
C          AUXILIARY PARAMETERS
C          ********************                                             200
C
C
C
C          SEMIMAJOR AXIS
C
      RTAI    =  1.D0 / (A * SRTHK)
      RTNAI   =  RTAI / XMEAN
      XNADR3  =  XMEAN / (RDA * RDA * RDA)
C
C          H AND K                                                         210
C
      BETAH   =  BETAUX * XH
      BETAK   =  BETAUX * XK
      BETAHN  =  BETAH / XMEAN
      BETAKN  =  BETAK / XMEAN
```

```
C
C           P AND Q
C
      TWOPQ   =  2.D0 * ONEPQI
      TWOIPQ  =  TWOPQ * RETRG                                          220
      PAUX    =  P * TWOIPQ
      QAUX    =  Q * TWOIPQ
C
C           PARTIAL DERIVATIVES OF POSITION IN ORBITAL REFERENCE
C           FRAME.
C
      DXDH  =  - BETAKN*VORB(1) + RTNAI*XORB(2)*VORB(2)
      DYDH  =  - BETAKN*VORB(2) - RTNAI*XORB(1)*VORB(2) - A
C
      DXDK  =    BETAHN*VORB(1) + RTNAI*XORB(2)*VORB(1) - A             230
      DYDK  =    BETAHN*VORB(2) - RTNAI*XORB(1)*VORB(1)
C
C           PARTIAL DERIVATIVES OF VELOCITY IN ORBITAL REFERENCE
C           FRAME.
C
      DVXDH  =    XNADR3 * (BETAK*XORB(1) - RTAI*XORB(2)*XORB(2))
     *         + RTNAI*VORB(2)*VORB(2)
      DVYDH  =    XNADR3 * (BETAK*XORB(2) + RTAI*XORB(1)*XORB(2))
     *         - RTNAI*VORB(1)*VORB(2)
C                                                                       240
      DVXDK  =  - XNADR3 * (BETAH*XORB(1) + RTAI*XORB(1)*XORB(2))
     *         + RTNAI*VORB(1)*VORB(2)
      DVYDK  =  - XNADR3 * (BETAH*XORB(2) - RTAI*XORB(1)*XORB(1))
     *         - RTNAI*VORB(1)*VORB(1)
C
C
C
C           ***************************
C           POSITION PARTIAL DERIVATIVES
C           ***************************                               250
C
C
C
C           SEMIMAJOR AXIS
C
  200 DPVDEQ (1,1)  =  X(1) / A
      DPVDEQ (2,1)  =  X(2) / A
      DPVDEQ (3,1)  =  X(3) / A
C
C           H                                                           260
C
      DPVDEQ (1,2)  =  DXDH*FAXIS(1) + DYDH*GAXIS(1)
      DPVDEQ (2,2)  =  DXDH*FAXIS(2) + DYDH*GAXIS(2)
      DPVDEQ (3,2)  =  DXDH*FAXIS(3) + DYDH*GAXIS(3)
C
C           K
C
      DPVDEQ (1,3)  =  DXDK*FAXIS(1) + DYDK*GAXIS(1)
      DPVDEQ (2,3)  =  DXDK*FAXIS(2) + DYDK*GAXIS(2)
```

```
      DPVDEQ (3,3)  =  DXDK*FAXIS(3) + DYDK*GAXIS(3)                    270
C
C           P
C
      FFAC  =    QAUX * XORB(2)
      GFAC  =  - QAUX * XORB(1)
      WFAC  =  - TWOPQ * XORB(1)
C
      DPVDEQ (1,4)  =  FFAC*FAXIS(1) + GFAC*GAXIS(1) + WFAC*WAXIS(1)
      DPVDEQ (2,4)  =  FFAC*FAXIS(2) + GFAC*GAXIS(2) + WFAC*WAXIS(2)
      DPVDEQ (3,4)  =  FFAC*FAXIS(3) + GFAC*GAXIS(3) + WFAC*WAXIS(3)    280
C
C           Q
C
      FFAC  =  - PAUX * XORB(2)
      GFAC  =    PAUX * XORB(1)
      WFAC  =    TWOIPQ * XORB(2)
C
      DPVDEQ (1,5)  =  FFAC*FAXIS(1) + GFAC*GAXIS(1) + WFAC*WAXIS(1)
      DPVDEQ (2,5)  =  FFAC*FAXIS(2) + GFAC*GAXIS(2) + WFAC*WAXIS(2)
      DPVDEQ (3,5)  =  FFAC*FAXIS(3) + GFAC*GAXIS(3) + WFAC*WAXIS(3)    290
C
C           MEAN LONGITUDE
C
      DPVDEQ (1,6)  =  V(1) / XMEAN
      DPVDEQ (2,6)  =  V(2) / XMEAN
      DPVDEQ (3,6)  =  V(3) / XMEAN
C
C
C
C           ***************************                               300
C           VELOCITY PARTIAL DERIVATIVES
C           ***************************
C
C
C
C           SEMIMAJOR AXIS
C
  300 VFAC  =  - .5D0 / A
C
      DPVDEQ (4,1)  =  VFAC * V(1)                                      310
      DPVDEQ (5,1)  =  VFAC * V(2)
      DPVDEQ (6,1)  =  VFAC * V(3)
C
C           H
C
      DPVDEQ (4,2)  =  DVXDH*FAXIS(1) + DVYDH*GAXIS(1)
      DPVDEQ (5,2)  =  DVXDH*FAXIS(2) + DVYDH*GAXIS(2)
      DPVDEQ (6,2)  =  DVXDH*FAXIS(3) + DVYDH*GAXIS(3)
C
C           K                                                           320
C
      DPVDEQ (4,3)  =  DVXDK*FAXIS(1) + DVYDK*GAXIS(1)
      DPVDEQ (5,3)  =  DVXDK*FAXIS(2) + DVYDK*GAXIS(2)
```

```
      DPVDEQ (6,3)  =  DVXDK*FAXIS(3) + DVYDK*GAXIS(3)
C
C           P
C
      FFAC  =    QAUX * VORB(2)
      GFAC  =  - QAUX * VORB(1)
      WFAC  =  - TWOPQ * VORB(1)                                        330
C
      DPVDEQ (4,4)  =  FFAC*FAXIS(1) + GFAC*GAXIS(1) + WFAC*WAXIS(1)
      DPVDEQ (5,4)  =  FFAC*FAXIS(2) + GFAC*GAXIS(2) + WFAC*WAXIS(2)
      DPVDEQ (6,4)  =  FFAC*FAXIS(3) + GFAC*GAXIS(3) + WFAC*WAXIS(3)
C
C           Q
C
      FFAC  =  - PAUX * VORB(2)
      GFAC  =    PAUX * VORB(1)
      WFAC  =    TWOIPQ * VORB(2)                                       340
C
      DPVDEQ (4,5)  =  FFAC*FAXIS(1) + GFAC*GAXIS(1) + WFAC*WAXIS(1)
      DPVDEQ (5,5)  =  FFAC*FAXIS(2) + GFAC*GAXIS(2) + WFAC*WAXIS(2)
      DPVDEQ (6,5)  =  FFAC*FAXIS(3) + GFAC*GAXIS(3) + WFAC*WAXIS(3)
C
C           MEAN LONGITUDE
C
      XFAC  =  - XNADR3
C
      DPVDEQ (4,6)  =  XFAC * X(1)                                      350
      DPVDEQ (5,6)  =  XFAC * X(2)
      DPVDEQ (6,6)  =  XFAC * X(3)
      RETURN
      END
```

```
      SUBROUTINE LUDCMP(A,N,NP,INDX,D)^M
c
c
c  LINPAK subroutine for decomposing an nxn matrix a.
c
c
      DOUBLE PRECISION A,SUM,DUM,VV,AAMAX,D
      PARAMETER (NMAX=100,TINY=1.0E-20)^M
      DIMENSION A(NP,NP),INDX(N),VV(NMAX)^M
      D=1.^M                                                        10
      DO 12 I=1,N^M
       AAMAX=0.^M
       DO 11 J=1,N^M
        IF (ABS(A(I,J)).GT.AAMAX) AAMAX=ABS(A(I,J))^M
11     CONTINUE^M
       IF (AAMAX.EQ.0.) PAUSE 'Singular matrix.'^M
       VV(I)=1./AAMAX^M
12    CONTINUE^M
      DO 19 J=1,N^M
       IF (J.GT.1) THEN^M                                           20
        DO 14 I=1,J-1^M
         SUM=A(I,J)^M
         IF (I.GT.1)THEN^M
          DO 13 K=1,I-1^M
           SUM=SUM-A(I,K)*A(K,J)^M
13        CONTINUE^M
          A(I,J)=SUM^M
         ENDIF^M
14      CONTINUE^M
       ENDIF^M                                                      30
       AAMAX=0.^M
       DO 16 I=J,N^M
        SUM=A(I,J)^M
        IF (J.GT.1)THEN^M
         DO 15 K=1,J-1^M
          SUM=SUM-A(I,K)*A(K,J)^M
15       CONTINUE^M
         A(I,J)=SUM^M
        ENDIF^M
        DUM=VV(I)*ABS(SUM)^M                                        40
        IF (DUM.GE.AAMAX) THEN^M
         IMAX=I^M
         AAMAX=DUM^M
        ENDIF^M
16     CONTINUE^M
       IF (J.NE.IMAX)THEN^M
        DO 17 K=1,N^M
         DUM=A(IMAX,K)^M
         A(IMAX,K)=A(J,K)^M
         A(J,K)=DUM^M                                               50
17      CONTINUE^M
        D=-D^M
        VV(IMAX)=VV(J)^M
```

```
      ENDIF^M
      INDX(J)=IMAX^M
      IF(J.NE.N)THEN^M
        IF(A(J,J).EQ.0.)A(J,J)=TINY^M
        DUM=1./A(J,J)^M
        DO 18 I=J+1,N^M
          A(I,J)=A(I,J)*DUM^M                                    60
18      CONTINUE^M
      ENDIF^M
19    CONTINUE^M
      IF(A(N,N).EQ.0.)A(N,N)=TINY^M
      RETURN^M
      END^M
```

```
      SUBROUTINE LUBKSB(A,N,NP,INDX,B)^M
c
c
c  LINPAK subroutine for carrying out back substitution.
c
c
      DOUBLE PRECISION A,B,SUM
      DIMENSION A(NP,NP),INDX(N),B(N)^M
      II=0^M
      DO 12 I=1,N^M                                                    10
        LL=INDX(I)^M
        SUM=B(LL)^M
        B(LL)=B(I)^M
        IF (II.NE.0)THEN^M
          DO 11 J=II,I-1^M
            SUM=SUM-A(I,J)*B(J)^M
11        CONTINUE^M
        ELSE IF (SUM.NE.0.) THEN^M
          II=I^M
        ENDIF^M                                                        20
        B(I)=SUM^M
12    CONTINUE^M
      DO 14 I=N,1,-1^M
        SUM=B(I)^M
        IF(I.LT.N)THEN^M
          DO 13 J=I+1,N^M
            SUM=SUM-A(I,J)*B(J)^M
13        CONTINUE^M
        ENDIF^M
        B(I)=SUM/A(I,I)^M                                              30
14    CONTINUE^M
      RETURN^M
      END^M
```